# Study on Using Reinforcement Learning for the Monotone Boolean Reconstruction

Hasmik Sahakyan[1,3*], Gyula Katona[2] and Levon Aslanyan[1]
[1]Institute for Informatics and Automation Problems of the National Academy of Sciences, P.Sevak street 1, Yerevan 0014, Armenia
[2]Alfréd Rényi lnstitute of Mathematics, Budapest, Reáltanoda u.13-15 PF 127, 1364, Hungary
[3]French University in Armenia, D.Anhaght street 10/1, Yerevan 0037, Armenia
E-mail: hsahakyan@sci.am, katona.gyula.oh@renyi.hu, lasl@sci.am
[*]Corresponding author

**Keywords:** Monotone Boolean functions, reinforcement learning, combinatorial optimization

**Received:** April 14, 2023

*This paper investigates the feasibility of using reinforcement learning to solve combinatorial optimization problems, in particular, the problem of query-based monotone Boolean function reconstruction. The monotone Boolean function reconstruction problem is a typical combinatorial problem that reconstructs the function unambiguously with a minimum number of queries about the value of the function at the defined points, based on the monotonicity of the function. The Shannon complexity of the problem is of the order of $2^n/\sqrt{n}$, and the solution algorithm relies on complex constructions, which also add complexity in the form of memory and time. Additionally, there are problems of partial reconstruction, e.g., in the mining of associative rules, which do not fit into the developed solution formats. This necessitates exploring heuristic domains to attract additional resources to solve the problem. To this end, all elements of reinforcement learning - environment, agent, policy, etc. - are designed, and both exact and approximate algorithms are given to perform the necessary structural data transformations, as well as to calculate the reward, the value, and other operational data of the algorithm. The focal point of the considerations is a subclass of monotone Boolean functions related to the well-known shadow minimization theorem of layer-by-layer characterized functions. Preliminary experiments have been started and they require follow-up intensive actions.*

*Povzetek: V raziskavi so avtorji preučili uporabo okrepljenega učenja za rekonstruiranje monotone Booleanove funkcije z minimalnim številom poizvedb. Predlagali so algoritme za točno in približno reševanje problema ter izvedli začetne simulacije.*

## 1 Introduction

Many problems with monotone Boolean functions (MBFs) appear not only in logical and physical level design of systems, but also in artificial intelligence models, computation learning theory, hypergraph theory, and other areas. MBFs are used to encode extremely important constructions in various combinatorial optimization problems; they provide a natural way to describe satisfiable subsets of finite constraint collections. Extreme points of MBFs correspond to maximal compatible subsets of constraints, such as sets of linear inequalities, closed sets of frequent elements in association rule searches, etc. A number of applications (e.g., wireless sensor networks, dead-end tests of tables, data mining [2,3]) are based on MBF optimization, where MBFs are represented not in direct form, but by using chains and anti-chains [29]. Other similar applications can be added to this description [6,8,17].

There are a number of effective tools and methods for analyzing MBFs, and new approaches are constantly being sought, investigated, and applied. Well-known open problems include the reconstruction of bounded classes of Boolean functions with randomization of queries and functions, and the use of cube-splitting and chain-splitting of the Boolean domain [2].

A well-known problem concerning MBFs is the identification problem – the recognition of an unknown MBF of $n$ variables by using membership queries. Hansel's algorithm [15], based on partitioning the binary cube into special non-intersecting chains, provides optimal reconstruction in the sense of Shannon complexity. In practical implementations, it is not necessary to build and store all chains in computer memory; according to [27,28,31], this can be done with simple procedures and an algebra that provides all the necessary answers to questions concerning the chains and queries mapped to these chains.

The problem of recognition of monotone Boolean functions with $n$ variables, for $n = 2, \cdots, 4$ is investigated in [4], where the authors compare the complexity of different types of optimal (relative to the depth or the number of realizable nodes) decision trees with hypotheses.

PAC (probably approximately correct) type learning algorithms of MBFs are considered in a number of papers [25,32]. An algorithm that learns any monotone Boolean function $f$ of $n$ variables, to any constant accuracy, under the uniform distribution, in time polynomial in $n$ and in the decision tree size of $f$ is achieved in [32].

In order to obtain solutions and approximations using MBF classes, it is necessary to reconstruct the function itself from examples/values of points. One of the key tools here is the well-known Kruskal-Katona theorem [13,14], which describes the exact optimal monotone construction for a given set of parameters. In this way, KK-MBF class of MBFs is formed, being a special and attractive class for recognition.

In recent years, machine learning (ML) has been seen as an additional technological resource for solving combinatorial optimization problems (CO) [18,22,33]. Numerous machine learning techniques have been used to solve various combinatorial optimization problems, such as reinforcement learning to train a deep Q-net [18] which is applied to the Traveling Salesman Problem (TSP) [16,26], graph convolution networks [22], pointer networks and learning-to-prune framework [33], decision trees and neural networks to classify discrete images with different structural properties using projection data [12], and others.

In this paper, we investigate the possibility of using ML to train an MBF recognition heuristic that uses structures of Boolean function classes and stepwise recognition based on membership queries, as well as similarity and approximation studies between MBF classes and fragments. Among machine learning techniques, we distinguish and apply Reinforcement Learning (RL) – an approach similar to the query-based recognition. In order to apply RL to combinatorial optimization problems, the problem must be modelled as a sequential decision-making process, where the agent interacts with the environment by performing a sequence of actions in order to find a solution. A fundamental question arises here, which requires a detailed analysis – whether it is possible to apply RL effectively for considered CO problem. Subjects for discussion include the compilation of acceptable sets of states and actions, possible and useful reward definitions, complexity aspects, and certainly, validation and estimation of approximations. These questions constitute the main topics of investigation of our present study. We propose an RL structure to solve the MBF recognition problem through a model (actions, states, and rewards) and an RL-MBF algorithm. Experiments involve simulations with a traditional RL algorithm SARSA, using instances of typical MBF classes.

The rest of the paper is organised as follows. Section 2 presents necessary definitions, preliminaries, and basic theoretical concepts of the RL and MBF recognition, respectively. Section 3 and 4 describe the proposed technique and model, RL structure and components for the MBF recognition, and RL MBF algorithm and data structures, respectively. Discussions and evaluation of the model are given in Section 5. The paper ends with some concluding remarks.

# 2 Preliminaries

## 2.1 Monotone Boolean function recognition

Let $B^n = \{(x_1, \cdots, x_n) \mid x_i \in \{0,1\}, i = 1, \cdots, n\}$ denote the set of vertices of the $n$-dimensional binary (unit) cube. Vertices of $B^n$ are obtained by assigning values to the binary variables $x_1, \cdots, x_n$. Let $\alpha = (\alpha_1, \cdots, \alpha_n)$ and $\beta = (\beta_1, \cdots, \beta_n)$ be two vertices of $B^n$. Then, $\alpha$ precedes $\beta$ (denoted as $\alpha \preccurlyeq \beta$) if and only if $\alpha_i \leq \beta_i$ for $1 \leq i \leq n$. If at the same time $\alpha \neq \beta$ then $\alpha$ strictly precedes $\beta$ (denoted as $\alpha \prec \beta$). $\alpha$ and $\beta$ are comparable if $\alpha \preccurlyeq \beta$ or $\beta \preccurlyeq \alpha$, otherwise, they are incomparable. A set of incomparable vertices in $B^n$ is also called a Sperner family. A (growing) chain is a sequence of vertices such that the $i$-th vertex in the sequence is obtained from the $(i-1)$-th vertex by replacing a "0" component with "1".

The *Hamming distance* between $\alpha$ and $\beta$ is the number of positions at which $\alpha_k \neq \beta_k$, $1 \leq k \leq n$.

For a given vertex $\alpha$ of $B^n$, we define two *intervals* of vertices as:

$$[\alpha, \tilde{1}] = \{\beta \in B^n | \alpha \preccurlyeq \beta \preccurlyeq \tilde{1}\}$$
$$[\tilde{0}, \alpha] = \{\beta \in B^n | \tilde{0} \preccurlyeq \beta \preccurlyeq \alpha\}$$

where $\tilde{1}$ and $\tilde{0}$ are vertices of $B^n$ with all components being 1s and 0s, respectively.

Boolean function $f: B^n \to \{0,1\}$ is called *monotone* if for every two vertices $\alpha, \beta \in B^n$, if $\alpha \prec \beta$ then $f(\alpha) \leq f(\beta)$. Vertices of $B^n$, where $f$ takes value "1" are called *units* or true points of the function; vertices, where $f$ takes value "0" are called *zeros* or false points of the function. $\alpha^1$ is a *lower unit* (or minimal true point) of the function if $f(\alpha^1) = 1$, and $f(\alpha) = 0$ for every $\alpha \in B^n$, such that $\alpha \prec \alpha^1$. $\alpha^0$ is an *upper zero* (maximal false point) of the function if $f(\alpha^0) = 0$, and $f(\alpha) = 1$ for every $\alpha \in B^n$ such that $\alpha^0 \prec \alpha$. $\min T(f)$ and $\max F(f)$ denote the sets of minimal true points and maximal false points, respectively. Obviously, $\min T(f)$ and $\max F(f)$ are Sperner families in $B^n$.

Formally, the work with MBFs started in 1987, with the issue of counting their number [9]. The first algorithmic and complexity-related considerations belong to [19], where, in particular, the valuable concept of *resolving subsets* was introduced. The final asymptotic estimate about the number of MBFs of $n$ variables was obtained in [20]. The technique on how to introduce and analyze MBFs, basically, is presented in [13,14,15,1,2,31,27,28,11].

The Hansel chain structure [15] was invented in 1966 and plays one of the central roles in MBF-related algorithmic techniques. The method is based on the partitioning of $B^n$ into $\binom{n}{\lfloor n/2 \rfloor}$ pairwise non-intersecting chains (Hansel chains), arranged symmetrically about the middle layers of $B^n$, which have the following properties: 1) The number of chains of the length $n-2p+1$ is $\binom{n}{p} - \binom{n}{p-1}$ for $0 \leq p \leq \lfloor n/2 \rfloor$; 2) For any three elements $\alpha^{i_1} \prec \alpha^{i_2} \prec \alpha^{i_3}$ of a chain of length $n-2p+1$, their

relative complement $\beta$ belongs to a chain of length $n - 2p - 1$.

The next valuable step to this was done by Tonoyan [31], who invented a set of simple procedures (chain algebra) that serve all the actual queries about Hansel chains, providing a technical solution to all the problems related to algorithms with Hansel chains, without constructing and keeping them in computer memory. In continuation, [27] presented a slightly modified and simplified version by using two tools: enumeration of all chains, and a procedure of finding the $i$-th vertex of the $j$-th chain. An optimised procedure is used to propagate newly found values to the chains, by a divide-and-conquer manner.

In the MBF recognition problem using membership queries, the goal is to determine an unknown MBF of $n$ variables using as few queries as possible. The function can be fully recognized by finding all its upper zeros (and/or lower units) [19]. The Shannon complexity of finding all upper zeros (lower units) of an arbitrary monotone Boolean function of $n$ variables is $\binom{n}{\lfloor n/2 \rfloor} + \binom{n}{\lfloor n/2 \rfloor + 1}$ [15].

General steps in any optimal in the sense of Shannon function algorithm of recognition of monotone Boolean function $f$ are the following:

1. Partitioning $B^n$ into Hansel chains;
2. Choosing a chain $j$ (usually, chains are in increasing order of length) and a vertex $\alpha$ on $j$ and querying $f(\alpha)$; On each chain, its "undetermined values" constitute an interval, and $\alpha$ is chosen as the middle point of the interval. Then the query response $f(\alpha)$ determines values in one half of the interval;
3. Propagating $f(\alpha)$ to other chains/vertices of $B^n$ whenever it is possible using the monotonicity of $f$;
4. Repeating steps 2-3 until $f$ is fully recognized (i.e., $f(\alpha)$ is obtained on all $\alpha \in B^n$).

Another recognition structure is used in [28]. For even $n$, $B^n$ is split according to two variables and the recognition in every sub-cube starts from its two middle layers. For odd $n$, firstly $B^n$ is split according to one variable, then as each sub-cube now has even size, the procedure for even sizes is applied. This provides optimal recognition of all MBFs in the sense of Shannon complexity. Unfortunately, while simple and attractive, this approach cannot be used in practical algorithms for arbitrary functions. Finally, it is worth mentioning the work [11] that considers not the Shannon complexity but the individual complexity of MBF given by its resolving set size. The core consideration is that for a given vertex $\alpha$, if $f(\alpha) = 0$, an upper-zero of $f$ can be found in no more than $n$ queries. The same holds with a lower-one when $f(\alpha) = 1$. This gives a recognition complexity bounded by $n$ times the resolving set size of $f$.

In general, tasks related to the recognition of MBFs may have different formulations. One task is to recognize a particular unknown function, knowing that it belongs to the class of MBFs or to one of its subclasses. Another task is to start with partial knowledge about the unknown function. One more case is when the number of queries is restricted by some number $K$ and the goal is to maximize the recognized part of the function.

Similar problems can be formulated for specific classes of Boolean functions. Examples of classes are as follows:

**KK-MBF** Kruskal-Katona MBF arises as a result of the shadow minimization theorem [21,13,14]. KK-MBF intersects the layers of the cube along their initial segments of lexicographic order. The complement of the KK-MBF area in $B^n$ has a similar property; it is related to the initial segments of co-lexicographic order. Partial KK-MBF is a partial MBF, where 0s of the function form initial segments of co-lexicographic order, and 1s form initial segments of lexicographic order on layers.

**Symmetric MBF** This is a trivial class of functions that take constant value on the cube layers. Examples are majority functions, parity functions, and others. Partial symmetric MBF is when the function is not determined on some layers of the cube.

**Threshold MBF** Functions are defined by a linear inequality of weighted sums of variables.

**Matroid MBF** Monotone Boolean function $f$ is called a matroid function if for each $\alpha, \beta \in \min T(f)$ with $\alpha_i = 1, \beta_i = 0$, there exists a coordinate $j$ with $\alpha_j = 0$, $\beta_j = 1$ such that vertex $\alpha'$, obtained from $\alpha$ by replacing $\alpha_i$ with 0 and $\alpha_j$ with 1, belongs to $\min T(f)$.

The combinatorial complexity of reconstruction in subclasses is not well studied. It is known that symmetric functions, with zeros and ones separated by two middle layers of the cube, are the most difficult functions for query-based reconstruction when only the monotonicity of the function is given. If it is known that the function belongs to the class of symmetric functions, the reconstruction can be done by $logn$ queries. The same function also belongs to $KK - MBF$, and the combinatorial complexity in this case does not exceed $nlogn$.

Other known and investigated classes of important MBFs, such as 2-monotonic positive functions $k$-tight functions, can be found in [7,24].

## 2.2 Reinforcement learning

Reinforcement learning (RL) [30,23] is an algorithmic implementation of the natural learning process in which there is a systematic and analytical interaction with the environment. The basic elements of RL are the environment and the learner agent that interacts with it. When interacting with the environment, the agent performs actions, and the environment responds through a reward. Accordingly, RL algorithms distinguish the

following main elements: *policy*, which is a mapping of environmental situations/states to actions and is the core of the reinforcement learning agent; *reward*, which characterizes the immediate reaction of the environment to the action; the *value* of state $s$ or the value of state-action pair $(s, a)$. The value function $V(s)$ is the expectation of future rewards when starting from state $s$ and following some policy $\pi$; and the action-value function $Q(s, a)$, which is the expectation of future rewards when starting from state $s$, taking action $a$ and following some policy $\pi$.

RL algorithms can be divided into policy-based and value-based methods. In the case of policy-based methods, the policy is approximated directly, whereas value-based methods focus on approximating a value function, which is a measure of policy quality for some state-action pair in the given environment. Value-based methods first compute the action-value function $Q(s, a)$ as the expected reward of the policy, given state $s$ and taking action $a$. The agent's policy then corresponds to the choice of action that maximizes $Q(s, a)$ for that state. The main difference between the value-based approaches is how to estimate $Q(s, a)$ accurately and efficiently.

Another important element of RL algorithms is the *model* of the environment, which may be unknown or partially known. The nature of RL elements - deterministic or stochastic, representation by function, table, or table being filled, presence of an environment model, nature of the applied policy, etc. – defines, in many respects, its mathematical model and problems of implementation.

RL is based on the concept of Markov decision-making processes (MDP) [5]. MDPs are structured from finite sets of actions, states, policies and a state transition model. A learner agent interacts with the environment in a sequence of steps in time (t): (i) the agent receives a representation of the current environment (some state); (ii) chooses and performs an action according to its policy; (iii) receives a reinforcement/reward; and (iv) enters a new state of the environment. The goal in MDP and reinforcement learning is to learn a policy $\pi$ that maximizes the expected sum of future rewards.

The epsilon-soft ($\varepsilon$-soft) is an example of action selection policy in RL, where the probability of all actions given a state $s$ is greater than some minimum value. The epsilon-greedy ($\varepsilon$-greedy) policy is a specific instance of an $\varepsilon$-soft, where the policy is applied according to the following equation:

$$\pi(s) = \begin{cases} a^* \; with \; probability \; 1 - \epsilon \\ a_a \; with \; probability \; \epsilon \end{cases}$$

where $\pi(s)$ is the decision policy for the current state $s$, $a^*$ is the best estimated action for the state $s$ at the current time, and $a_a$ is a random action selected with probability $\epsilon$ [30].

In practice it is rare that an agent learns optimal policy; only an approximation can be achieved in various learning algorithms. The available memory is an important constraint; large amount of memory is often required to build up approximations of value functions, policies, and models. In tasks with small, finite state sets, it is possible to form these approximations using learning tables with one entry for each state (or state–action pair). In many cases of practical interest, however, there are far more states than could possibly be entries in a table. In these cases the functions must be approximated.

The online nature of reinforcement learning allows optimal policies to be approximated in ways that prioritize learning to make good decisions for frequently encountered states, at the expense of less effort for infrequently encountered states. This is one key property that distinguishes reinforcement learning from other approaches to approximately solving MDPs.

SARSA, a counterpart of Q-learning approach [30], is one of the common approximate value-based methods. It is based on temporal difference learning, where update estimates are partially based on other learned estimates, without waiting for a final outcome. SARSA iteratively makes the following update of the action-value on transition to the next state:

$$Q_{t+1} \coloneqq Q_t(s, a) + \alpha[r(s, a) + \gamma Q_t(s', a') - Q_t(s, a)]$$

where $s$ is the state and $a$ is the action at the current step $t$, respectively; $s'$ is the state and $a'$ is the action at the next step $t + 1$. $Q_t(s, a)$ is the action-value at time $t$ for the state-action pair $(s, a)$. $Q_{t+1}$ is the update at time $t + 1$ by performing action $a$ in state $s$; $r(s, a)$ is the reward received for the pair $(s, a)$; $\alpha$ and $\gamma$ are the parameters, where $\gamma$ is the scalar discount factor, $0 < \gamma \leq 1$, that describes the agent preference between current and future reward; $\alpha$ is the learning rate, which controls overlap speed of new information.

In general, agent–environment interaction is naturally broken down into a sequence of separate episodes, and each action affects only the finite number of rewards subsequently received during the episode. The time steps of each episode are numbered starting anew from zero.

**SARSA Algorithm**

Algorithm parameters: step size $\alpha \in (0,1]$, small $\epsilon > 0$;
Initialize $Q(s, a)$ arbitrary for each pair $(s, a)$; except that $Q(terminal\ state, a)$ must be 0 for all $a$;
Loop for each episode
  Observe state $s$;
  Select $a$ using policy derived from $Q$, e.g., $\epsilon$ –greedy;
  Repeat
    Take action $a$;
    Receive reward $R(s, a)$;
    Observe the next state $s'$;
    Select the new action $a'$ using policy derived from $Q$, e.g., $\epsilon$ –greedy;
    Update $Q(s, a)$ with
    $Q(s, a) \coloneqq$
    $Q(s, a) + \alpha[R(s, a) + \gamma Q(s', a') - Q(s, a)]$;
    $s \coloneqq s', a \coloneqq a'$
  Until $s$ is terminal

# 3 RL components for MBF recognition

The idea of using RL for MBF recognition is to replace the difficult task of determining the next vertex of the

binary cube $B^n$ in combinatorial algorithms (to query the function value) with using examples, parameters learning and mimicking the best behaviour. The whole RL framework is composed of combinatorial preparations and learning heuristics. To understand the role of combinatorial and heuristic procedures in the overall RL structure, consider the data structures of the SARSA algorithm given in Figure 1.

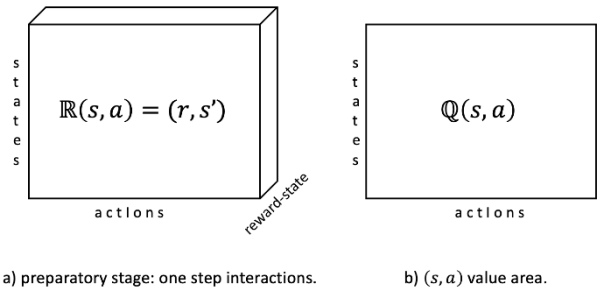
Data in part a) of **Error! Reference source not found.** is algorithmic in nature. We call this the Reward table. In the preparatory stage of the problem, it is necessary to define adequate reward values for all state-action pairs $(s, a)$ (assuming that we have the state and action sets). The reward values will not be changed during the whole performance. The new state $s'$ is also definite, in the sense that being in state $s$ and performing action $a$ will lead to $s'$, defined by deterministic or stochastic transition rules. In this part, we will intensively use technologies largely implemented in algorithmic studies of MBF recognition. The table $\mathbb{R}$ can be completely or partially filled, containing only the values that have already been calculated by that moment.

Often, also in MBF recognition, the task of filling $\mathbb{R}$ is solved by searching or using other discrete, combinatorial considerations.

Data in part b) is in the operative domain controlled by the learning process. Initially, this data is initialized with zero values and then learned step-by-step. The learning is based on the learning method (given in [30] for SARSA), which is also based on the learning policy.

On the one hand, policy uses the table $\mathbb{Q}$, the part of it that is already updated, and on the other hand, it is extended with a "planning" step that assigns some arbitrary actions randomly. This is a kind of balance between known and unknown rewards. SARSA, having the current state $s$, selects action $a$ using the policy derived from $\mathbb{Q}$, obtains $(r, s')$ from the reward table, and selects the next action $a'$ by the policy, and on this basis learns and updates the new value of $\mathbb{Q}$.

We conclude: the learning part of SARSA is universal; it works with the policy function $\pi$ and the value table $\mathbb{Q}$, and these areas as well as the learning procedure are independent of the subject domain under consideration. The complementary part related to the reward table $\mathbb{R}$ defines one-step-rewards, which in our case is a combinatorial computational problem. Then, the concept of learning is to integrate these one-step-rewards into longer chains of predictions, thus intending to achieve the learning goal. For needs of learning over MBF classes, it is now clear that we will apply the combinatorial technique developed in this area.




a) preparatory stage: one step interactions. b) $(s, a)$ value area.

Figure 1: Basic data structures used in typical SARSA

## 3.1 MDP model on MBF reconstruction

In a typical MBF recognition problem using membership queries, the goal is to determine an unknown MBF $f$ of $n$ variables by as few queries as possible. We can also consider the problem of determining whether an unknown Boolean function belongs to the class of monotone functions. Another case is when the number of queries is limited to some number $K$ and the goal is to maximize the recognized part of the function.

In the following, we will point out a number of individual problems of this type that may arise in RL MBF applications, and their mathematical formulations. Suppose that the vertices of $B^n$ are enumerated. Let $x_j$ denote binary variables such that $x_j$ takes value 1 if the $j$-th vertex of $B^n$ is chosen to ask the value of the function, and 0 otherwise. $c_j$ expresses the informativeness of the $j$-th vertex, i.e., the number of new evaluations of the function obtained by propagating the value of the $j$-th vertex.

The formalism of these steps, depending on the particular objectives, may be given as follows:

1) $min \sum_j x_j$, subject to $\sum_j c_j = 2^n$, this is the case when the function is to be entirely recovered and the goal is to minimize the number of queries;
2) $min \sum_j x_j$, subject to $\sum_j c_j \geq N$, this is the case when a certain part of the function is to be recovered by the minimum number of queries;
3) $max \sum_j c_j$, subject to $\sum_j x_j = K$, this is the case when the number of queries is restricted and the goal is to maximize the recognized part of the function.

Note that the values of $c_j$ become known only after the vertices are selected. But regardless of the order of selection, $\sum_j c_j$ will have the same value at the end.

In order to apply RL to MBF recognition we need to reformulate it in terms of MDP, i.e., define the environment, states, actions, rewards, transition, and policy.

### 3.1.1 Environment

The basis of the RL environment for the MBF reconstruction considered in this paper, is the $n$-dimensional binary cube, its vertex set $B^n$, and the related sub-cube structure. The environment may need additional tools /known concepts/ such as Sperner systems, Hansel chains, Hamming distance, lexicographic order, and others.

### 3.1.2 States

The states represent some /partial/ knowledge about the solution, such us nested parts of the looked-for MBF, which we call *fragments*. An initial fragment can be the empty set, or we can start with some non-empty fragment. Thus, formally, each state $s$ is a partially defined monotone Boolean function $f^s$, given through a pair $(T_{f^s}, F_{f^s})$ of some sets of true and false vertices.

An agent must travel through the remaining (undetermined) vertices of the cube and decide (according to the policy) which vertex to choose as a query. Thus, actions are vertices chosen to ask the value of the function.

### 3.1.3 Rewards

Since the agent's instrument of interaction with the environment is a query $\alpha \in B^n$, the response of the environment should characterize how successful the query is, having a predetermined partial monotone Boolean function available when the query is performed. Thus, the reward will express the "informativeness" of the chosen vertex-action $\alpha$ in the sense of spreading the value $f(\alpha)$ to other vertices due to the monotonicity of $f$.

### 3.1.4 Transition/update

Depending on the chosen vertex and the function value, new state $s'$ will be determined, which is a new partially defined function $f^{s'}$ given through a new pair $(T_{f^{s'}}, F_{f^{s'}})$, such that $T_{f^s} \subseteq T_{f^{s'}}$ and $F_{f^s} \subseteq F_{f^{s'}}$.

### 3.1.5 Terminal state

The process will be continued until the termination criterion is reached, which according to the defined objectives is as follows:

1) $f$ is completely recovered,
2) $f$ is recovered by a certain percentage,
3) a certain number of steps are done.

### 3.1.6 Policy

Environment models, when known, are used for planning, by which we mean an arbitrary approach of deciding to apply actions by considering possible future situations before they actually occur. Methods for solving reinforcement learning problems that use models and planning are called model-based methods, in contrast to methods without models that learn by trial and error.

The goal of all reinforcement learning algorithms is to find a policy (the way of choosing actions), which would consistently allow the agent to gain a lot of rewards. For MBF recognition, the goal is to find a policy to get as many known values of the function as possible (ideally, this will be the whole function).

Initially, we will start with the following: being in any state $s$, actions are chosen from the area $B^n \setminus (T_{f^s} \cup F_{f^s})$, using $\varepsilon$-greedy method. According to MBFs structure, vertices of $\bigcup_{\alpha \in T_{f^s}} [\alpha, \tilde{1}]$ are true vertices, and vertices of $\bigcup_{\alpha \in F_{f^s}} [\tilde{0}, \alpha]$ are false vertices of $f$. It is then natural to exclude this part from further consideration. Thus, another policy may be: being in any state $s$, choose actions from the area $B^n \setminus (\bigcup_{\alpha \in T_{f^s}} [\alpha, \tilde{1}] \cup \bigcup_{\alpha \in F_{f^s}} [\tilde{0}, \alpha])$ using $\varepsilon$-greedy method.

Moreover, special policies may be defined that exploit structures of MBF, and are based on Hamming distances between selected points in $T_{f^s}$ and $F_{f^s}$.

## 3.2 Initial fragments, policies, updates, and rewards

We consider three types of initial fragments.

### 3.2.1 Partially defined monotone Boolean functions given by Sperner families

The input fragment (corresponding to the initial state $s$) is given through a pair of feasible/compatible Sperner families $(T_{f^s}, F_{f^s})$, one for the lower units, and one for the upper zeros of a partially defined monotone Boolean function $f^s$, as shown in Figure 2 a).

**Policy 1 -** actions are selected in $B^n \setminus (T_{f^s} \cup F_{f^s})$ using $\varepsilon$-greedy method.

**Actions**

Action $a$ in a given state $s$ is a vertex $\alpha$ in $B^n \setminus (T_{f^s} \cup F_{f^s})$, chosen to ask the value $f(\alpha)$.

**New states**

Depending on $f(\alpha)$ the following operations are to be performed:

For $f(\alpha) = 1$

i. if $\alpha$ is not comparable with any vertex of $T_{f^s}$, then $\alpha$ is added to $T_{f^s}$, thus the new state $s'$ is defined by the pair $(T_{f^{s'}}, F_{f^{s'}})$, where
$$T_{f^{s'}} = T_{f^s} \cup \{\alpha\} \text{ and } F_{f^{s'}} = F_{f^s}$$
ii. if $\alpha$ precedes some vertex/vertices $\beta$ of $T_{f^s}$, then $\alpha$ is added to $T_{f^s}$ and all vertices greater than $\alpha$ are removed from $T_{f^s}$, thus
$$T_{f^{s'}} = (T_{f^s} \cup \{\alpha\}) \setminus \{\beta | \beta \in T_{f^s} \text{ and } \beta \succ \alpha\}$$
iii. if there is a vertex $\beta$ of $T_{f^s}$ that precedes $\alpha$, then the partial solution /state/ is not changed,
$$T_{f^{s'}} = T_{f^s} \text{ and } F_{f^{s'}} = F_{f^s}$$

For $f(\alpha) = 0$

iv. if $\alpha$ is not comparable with any vertex of $F_f$, then $\alpha$ is added to $F_{f^{s'}}$, thus
$$F_{f^{s'}} = F_{f^s} \cup \{\alpha\}, T_{f^{s'}} = T_{f^s}$$

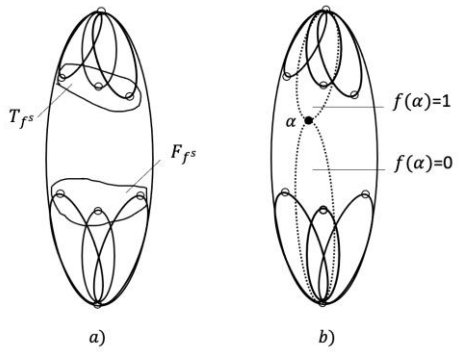



Figure 2: Partial MBF given by Sperner systems $T_{f^s}$ and $F_{f^s}$

v. if there is a vertex/vertices $\beta$ of $F_{f^s}$ that precedes $\alpha$, then $\alpha$ is added to $F_{f^s}$ and all vertices that precede $\alpha$ are removed from $F_{f^s}$, thus $F_{f^{s\prime}} = F_{f^s} \cup \{\alpha\} \backslash \{\beta | \beta \in F_{f^s} \; and \; \beta \prec \alpha\}$

vi. if $f$ $\alpha$ precedes some vertex $\beta$ of $F_{f^s}$, then the partial solution /state/ is not changed, $T_{f^{s\prime}} = T_{f^s}$ and $F_{f^{s\prime}} = F_{f^s}$.

An illustration is given in Figure 2 b).

**Rewards**

We will use approximate reward calculations based on Hamming distances between the vertex $\alpha$ and the vertices of $T_{f^s}$ and $F_{f^s}$.

**Reward 1**

The reward is 0 for the cases iii and vi, otherwise, it is the average Hamming distance, when averaged over all distances from $\alpha$ to the vertices of $T_{f^s}$ and $F_{f^s}$:

$$R = \frac{\sum_{\gamma \in (T_{f^s} \cup F_{f^s})} \rho(\alpha, \gamma)}{\left|T_{f^s} \cup F_{f^s}\right|}$$

**Reward 2**

The reward is 0 for the cases iii and vi, otherwise, it is the average Hamming distance, when the average is over the two farthest distances from $\alpha$ in $T_{f^s}$ and $F_{f^s}$:

$$R = \frac{\max\limits_{\gamma \in T_{f^s}} \rho(\alpha, \gamma) + \max\limits_{\gamma \in F_{f^s}} \rho(\alpha, \gamma)}{2}$$

### 3.2.2 Partially defined monotone Boolean functions given by bunch of intervals

The input fragment (corresponding to the initial state $s$) is given through a pair of bunches of intervals $(\cup_{\alpha \in T_{f^s}}[\alpha, \tilde{1}], \cup_{\alpha \in F_{f^s}}[\tilde{0}, \alpha])$; initially, $T_{f^s}$, $F_{f^s}$ are feasible/compatible Sperner families, one for the lower units, and one for the upper zeros of a partially defined monotone Boolean function $f^s$.

**Policy 2 -** actions are selected in $B^n \backslash (\cup_{\alpha \in T_{f^s}}[\alpha, \tilde{1}] \cup \cup_{\alpha \in F_{f^s}}[\tilde{0}, \alpha])$ using $\varepsilon$-greedy method.

We have two options:

1. Keep all intervals $\cup_{\alpha \in T_{f^s}}[\alpha, \tilde{1}]$ and $\cup_{\alpha \in F_{f^s}}[\tilde{0}, \alpha])$, - this requires extra memory;
2. Exclude all vertices greater than any vertex of $T_{f^s}$, and lower than any vertex of $F_{f^s}$ – this requires extra computations.

**Actions**

Action $a$ in given state $s$ is a vertex $\alpha$ in $B^n \backslash (\cup_{\alpha \in T_{f^s}}[\alpha, \tilde{1}] \cup \cup_{\alpha \in F_{f^s}}[\tilde{0}, \alpha])$, chosen to ask the value $f(\alpha)$.

**New states**

Depending on $f(\alpha)$ the following operations are to be performed ($T_{f^{s\prime}}$ and/or $F_{f^{s\prime}}$ may not be Sperner families):

i. if $f(\alpha) = 1$ then $T_{f^{s\prime}} := T_{f^s} \cup \{\alpha\}$ and $F_{f^{s\prime}} := F_{f^s}$,

ii. if $f(\alpha) = 0$ then $F_{f^{s\prime}} := F_{f^s} \cup \{\alpha\}$ and $T_{f^{s\prime}} := T_{f^s}$,

the new state is $(\cup_{\alpha \in T_{f^{s\prime}}}[\alpha, \tilde{1}], \cup_{\alpha \in F_{f^{s\prime}}}[\tilde{0}, \alpha])$.

**Rewards**

**Reward 1**

$$R = \left|(\cup_{\alpha \in T_{f^{s\prime}}}[\alpha, \tilde{1}] \backslash \cup_{\alpha \in T_{f^s}}[\alpha, \tilde{1}]) \cup (\cup_{\alpha \in F_{f^{s\prime}}}[\tilde{0}, \alpha]) \backslash (\cup \cup_{\alpha \in F_{f^s}}[\tilde{0}, \alpha])\right|$$

**Reward 2**

$$R = \frac{\left|(\cup_{\alpha \in T_{f^{s\prime}}}[\alpha, \tilde{1}] \backslash \cup_{\alpha \in T_{f^s}}[\alpha, \tilde{1}]) \cup (\cup_{\alpha \in F_{f^{s\prime}}}[\tilde{0}, \alpha]) \backslash (\cup \cup_{\alpha \in F_{f^s}}[\tilde{0}, \alpha])\right|}{\left|B^n \backslash (\cup_{\alpha \in T_{f^s}}[\alpha, \tilde{1}] \cup \cup_{\alpha \in F_{f^s}}[\tilde{0}, \alpha])\right|}$$

$$= \frac{\left|(\cup_{\alpha \in T_{f^{s\prime}}}[\alpha, \tilde{1}] \backslash \cup_{\alpha \in T_{f^s}}[\alpha, \tilde{1}])\right| + \left|(\cup_{\alpha \in F_{f^{s\prime}}}[\tilde{0}, \alpha]) \backslash (\cup \cup_{\alpha \in F_{f^s}}[\tilde{0}, \alpha])\right|}{2^n - \left|\left(\cup_{\alpha \in T_{f^s}}[\alpha, \tilde{1}]\right)\right| - \left|\cup_{\alpha \in F_{f^s}}[\tilde{0}, \alpha]\right|}$$

**Reward 3**

For obtaining an approximate value for the reward, we can apply Bonferroni inequalities [35,36] for approximating the value of corresponding inclusion-exclusion formula. For example, $R = \left|[\alpha', \tilde{1}] \backslash \cup_{\alpha \in T_{f^s}}[\alpha, \tilde{1}]\right|$ can be approximated in the following way.

We suppose that $h_1, h_2, \ldots, h_q$ are the smallest vertices of $T_{f^s}$ that are greater than $\alpha'$.

Denote $H_p = [h_p, \tilde{1}]$ for $p = 1, \cdots, q$. The exact reward value can be expressed by the inclusion-exclusion formula:

$$R = 2^{|\alpha\prime|} - \left|\cup_{p=1}^{q} H_p\right| = 2^{|\alpha\prime|} - \sum_p |H_p|$$

$$+\sum_{i<j}|H_i \cap H_j| - \sum_{i<j<k}|H_i \cap H_j \cap H_k| + \cdots (-1)^q |H_1 \cap H_2 \cap \ldots \cap H_q|,$$

where $|\alpha'|$ denotes the size of the interval $[\alpha', \tilde{1}]$, and $2^{|\alpha'|}$ is the number of vertices in $[\alpha', \tilde{1}]$.

The sum of some first terms in the formula is alternately an upper bound and a lower bound, which is the key for approximations.

### 3.2.3 Points on chains

The input fragment is composed as follows: split $B^n$ into Hansel chains (Figure illustrates the Hansel chains in $B^5$), and compose a list $L = (l_1, l_2, \cdots, l_{\binom{n}{\lfloor n/2 \rfloor}})$ of their lengths; for example, $L = (6,4,4,4,4,2,2,2,2,2)$ for $B^5$. Take at random two numbers in $[1, l_j]$ for each chain $j$. The smaller number will indicate the initial maximal false point, and the larger number will indicate the initial minimal true point of the chain.

Compose $T_{f^s}$ and $F_{f^s}$ from the initial minimal true and maximal false points of the chains, respectively. Let $(\alpha_j, \beta_j)$, $\alpha_j \prec \beta_j$, be the current maximal false and minimal true vertices of the $j$-th chain. Compose intervals $[\alpha_j, \beta_j]$ on the chains and call them "uncertainty intervals" of the chains.

**Policy 3 -** actions are chosen in $\bigcup_j [\alpha_j, \beta_j]$ using $\varepsilon$-greedy method.

**Actions**

Action $a$ in given state $s$ is a vertex $\gamma$ in $\bigcup_j [\alpha_j, \beta_j]$ chosen to query the value $f(\gamma)$.

**New states**

Let $\gamma$ belongs to $[\alpha_j, \beta_j]$

i. if $f(\gamma) = 1$ then $\gamma$ replaces the true point $\alpha_j$ of the chain, $\alpha'_j := \gamma$, resulting a new smaller uncertainty interval $[\alpha'_j, \beta'_j]$, $\beta'_j = \beta_j$. Moreover, $f(\gamma)$ can be spread to the vertices of the uncertainty intervals of all chains, thus replacing their true points (for simplicity we may omit this);
ii. if $f(\gamma) = 0$ then $\gamma$ replaces the false point $\beta_j$ of the chain, $\beta'_j := \gamma$, resulting a new smaller uncertainty interval $[\alpha'_j, \beta'_j]$, $\alpha'_j = \alpha_j$. Moreover, $f(\gamma)$ can be spread to the vertices of the uncertainty intervals of all chains, thus replacing their false points (for simplicity we may omit this).

We note that any optimal MBF recognition algorithm that works with Hansel chains chooses the middle points of the intervals to ask the value of the function. But the goal here is to determine the most informative point for a given function depending on its structure.

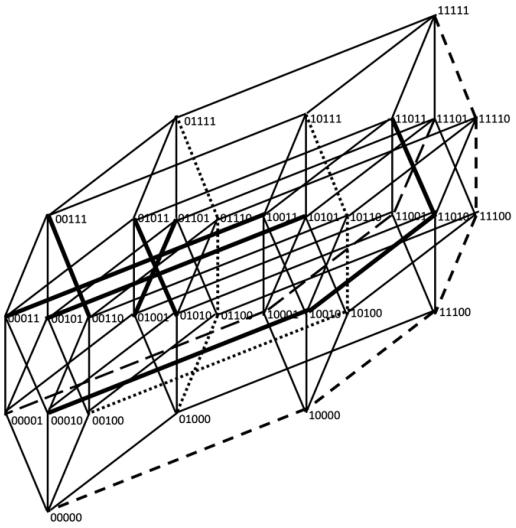



Figure 3: Hansel chains in $B^5$

**Rewards**

The reward for action $\alpha$ in state $s$ depends on the "informativeness" of the vertex.

**Reward 1**

The reward is the difference between the summary lengths of the new and previous uncertainty intervals:

$R = \left| \bigcup_j [\alpha_j, \beta_j] \setminus \bigcup_j [\alpha'_j, \beta'_j] \right|$

**Policy 4 -** actions are chosen starting from the largest intervals using $\varepsilon$-greedy method.

**Actions**

An action $a$ in a given state $s$ is a vertex $\gamma$ in the largest $[\alpha_j, \beta_j]$ chosen to ask the value $f(\gamma)$.

New states and rewards can be defined as for the previous case.

# 4 RL MBF algorithm

In this section we describe an RL algorithm for the case when states are defined by Sperner families, and the objective is to maximize the recognized part of the function, when the number of queries is restricted by some natural number $K$,

$max \sum_j c_j$
subject to $\sum_j x_j = K$.

The proposed algorithm is based on SARSA method. The policy $\pi$ and the reward function $R$ can be any of those defined in the previous section.

**RL-MBF algorithm is composed of several parts:**

Set parameters $\alpha, \gamma$, policy $\pi$, reward function $R$, number of episodes N (episodes will be explained in Section 4.2)
Initialize RL-MBF variables and constants, including the cube size $n$, and integers $k, l, 0 \le k, l \le n$ to characterise Sperner families

Generate initial state $s$ (consists of two Sperner families, one is for $k$ minimal true points, and another is for $l$ maximal false points)
Get feasibility area (this is "feasible" part of $B^n$ for selecting valid query vertices, thus, excluded initial two Sperner families, or Sperner families along with the corresponding intervals, according to the policy $\pi$)
Create root vertex in Q-graph according to the initial state $s$;
Repeat for each episode
  Select vertex $\alpha$ /action $a$/ in the feasibility area according to the policy $\pi$
  Repeat
    Query the value $f(\alpha)$
    Get new state $s'$ (according to the value $f(\alpha)$ and the policy $\pi$)
    Calculate Reward for $(s, a)$ (according to Reward function $R$)
    Update feasibility area (remove the "difference" according to the policy $\pi$)
    Select new vertex $\alpha'$ /action $a'$/ in the feasibility area according to the policy $\pi$
    Update Q-table (update Q-graph using $s, a, s', a'$ according to SARSA)
    Calculate cumulative reward
    $s := s'$
    $a := a'$
  until $K$ steps are done
until N episodes are done

## 4.1 Functions used in RL-MBF

In this section we describe some of the functions used in RL-MBF algorithm.

Let $\alpha = (a_1, \cdots, a_n)$ and $\beta = (b_1, \cdots, b_n)$ be vertices of $B^n$. $\alpha$ precedes lexicographically $\beta$ if either there exists an integer $k$, $1 \le k \le n$, such that $a_k < b_k$ and $a_i = b_i$ for $i < k$, or $\alpha = \beta$. The vectors of $B^n$ are in a lexicographic order in the sequence $\alpha_0, \alpha_1, \cdots, \alpha_{2^n-1}$ if $\alpha_i$ precedes lexicographically $\alpha_j$, for $0 \le i < j \le 2^n - 1$.

The *serial number* of the vertex $\alpha = (a_1, \cdots, a_n)$ is the natural number $a_1 2^{n-1} + a_2 2^{n-2} + \cdots + a_n 2^0$, whose binary representation is $a_1 a_2 \cdots a_n$. When the vectors of $B^n$ are in a lexicographic order, their serial numbers form the sequence $0,1, \cdots, 2^n - 1$.

**Generate initial state $s$ and Get feasibility area (for Policy 1)**

$T_{f^s} := \emptyset$; $F_{f^s} := \emptyset$; $F = \{0,1, \cdots, 2^n - 1\}$; $k$, $l$;
$t := 0$;
repeat
  $r$:= random number in $F$;
  find vertex $\alpha_r$, with the serial number $r$;
  if $\alpha_r$ is not comparable with any vertex of $T_{f^s}$ then
    $T_{f^s} := T_{f^s} \cup \{\alpha_r\}$;
    $F := F \backslash \{r\}$;
    $t = t + 1$;
until $t = k$
$t := 0$;
repeat
  $r$:= random number in $F$;
  find vertex $\alpha_r$ with the serial number $r$;
  if $\alpha_r$ is not comparable with any vertex of $F_{f^s}$ and is not greater of any vertex of $T_{f^s}$ then
    $F_{f^s} := F_{f^s} \cup \{\alpha_r\}$;
    $F := F \backslash \{r\}$;
    $t = t + 1$;
until $t = l$
$s := T_{f^s} \cup F_{f^s}$;

**Generate initial state $s$ and Get feasibility area (for Policy 2)**

$\hat{T}_{f^s} := \emptyset$; $\check{F}_{f^s} := \emptyset$; $F = \{0, \cdots, 2^n - 1\}$; $k$, $l$;
repeat
  $r$= random number in $F$;
  find vertex $\alpha_r$ with the serial number $r$;
  $\hat{T}_{f^s} := \hat{T}_{f^s} \cup [\alpha_r, \tilde{1}]$;
  $F := F \backslash \{serial\ numbers\ of [\alpha_r, \tilde{1}]\}$;
until $k$ steps are done

repeat
  $r$:= random number in $F$;
  find vertex $\alpha_r$ with the serial number $r$;
  $\check{F}_{f^s} := \check{F}_{f^s} \cup [\tilde{0}, \alpha_r]$;
  $F := F \backslash \{serial\ numbers\ of\ [\tilde{0}, \alpha_r]\}$;
until $l$ steps are done
$s := \hat{T}_{f^s} \cup \check{F}_{f^s}$;

**Get new state $s'$ and Update feasibility area (for Policy 2)**

If $f(\alpha) = 1$
  then
    $\hat{T}_{f^{s'}} := \hat{T}_{f^s} \cup [\alpha, \tilde{1}]$;
    temp:= $F \backslash \{serial\ numbers\ of\ [\alpha, \tilde{1}]\}$;
    diff:= $F$\temp; $F$ :=temp;
  else
    $\check{F}_{f^{s'}} := \check{F}_{f^s} \cup [\tilde{0}, \alpha]$;
    temp:= $F \backslash \{serial\ numbers\ of\ [\tilde{0}, \alpha]\}$;
    diff:= $F$\temp;
    $F$ :=temp;

RL-MBF algorithm, as well as functions used can be described in a similar way for the case, where states are defined by points on chains.

## 4.2 Data structure for Q-table

As mentioned earlier, agent–environment interaction is broken down into separate episodes, and each action affects only the finite number of rewards subsequently received during the episode.

In the case of MBF recognition, each episode starts with the same initial fragment (e.g. two initial Sperner families along with the corresponding intervals) and has fixed length $K$. Theoretically, an action can be any vertex of $B^n$, excluding the vertices of the initial fragment, and a

state can be any partial monotone Boolean function containing the initial fragment.

Within a single episode, a sequence of steps leads to a sequence of states that correspond to a series of nested partial monotone Boolean functions. Thus, "feasibility" area for choosing actions becomes more restricted from state to state. In terms of the reward table $R$, states are assigned to the rows of the table, and actions are assigned to the columns. Each state has its area of "feasible" actions (a subset of columns). At each step of an episode, being in a state $s$ and performing an action $a$, if $a$ is a new action for the state $s$, then cell $(s, a)$ is filled with the computed reward value $R(s, a)$, otherwise, the cell is already filled with the reward value. For the first episode, $R$ is initially empty, and at each step exactly one cell is filled.

Regarding the state-action table $Q$, at each step of an episode, being in state $s$ and performing action $a$, if $a$ is a new action for the state $s$, then cell $(s, a)$ is assigned 0, otherwise $Q(s, a)$ is updated according to the method used (e.g., SARSA). For the first episode, the table is initially empty.

Given the features of MBF, i.e. the nested structure of states and the restriction of the range of feasible actions from state to state, instead of storing Reward- and Value-tables we create an oriented weighted graph $Q$-graph as follows.

States are assigned to vertices and state-action pairs are assigned to edges. One vertex is mentioned as the root vertex that corresponds to the initial state; the root has no incoming edges.

Each episode is an oriented path in $Q$-graph, starting from the root and having length $K$. Each edge $(s, a)$ is assigned two weights; one is the reward $R(s, a)$, the other is the state-action value $Q(s, a)$. For the first episode, the entire path is newly created; $R(s, a)$ are computed according to the reward function, and $Q(s, a)$ are assigned 0, then $Q(s, a)$ are updated according to the SARSA formula.

Obviously, paths for different episodes can overlap (i.e., the same state can be reached by different sequences of actions), possibly, making cycles.

In the current episode, being in some state $s$ and choosing action $a$, we search among the outgoing edges from $s$; if edge $(s, a)$ already exists, then the pair has already met, and $R(s, a)$ and $Q(s, a)$ have been calculated.

If edge $(s, a)$ does not exist, we first look for vertex $s'$, if it exists we add edge $(s, a)$ from $s$ to $s'$, if it does not exist then we create new vertex $s'$ and add edge from $s$ to $s'$. Then we calculate $R(s, a)$ according to the reward function, assign $Q(s, a)$ to 0, and then update it according to SARSA formula.

In order to organize the search process for vertex $s$ and edges originating from $s$, as well as Reward and Q tables, we introduce the following structure.

We create an array "Vertex_list", each element $v$ of which has its name "$v$.name" and its set of outgoing edges "$v$.edge_list". "$v$.name" keeps states, and "$v$.edge_list" keeps all outgoing edges from $v$. Initially, "Vertex_list" consists of the root vertex $v_0$; "$v_0$.name" is the initial state, "$v_0$.edge.list" is empty.

Each element "edge: of "$v$.edge_list", in its turn, has its own name "edge.name", which is the chosen vertex-action, its reward "edge.reward", and its Q-value "edge.value".

Update is organized in the following way.

**Update Q($s, a, s', a'$)**

If $a$ does not exist in "$s$.edge_list", and $s'$ does not exist in "Vertex_list", then create a new graph vertex $s'$ in "Vertex_list" with "$s'$.name"= $s'$, and add outgoing edge $a$ from $s$ to $s'$ by adding $(s, a)$ into the "$s$.edge_list" of $s$; with "edge.value" 0; and "edge.reward" - calculated according to the reward function.

If $a$ does not exist in "$s$.edge_list", but $s'$ exists in "Vertex_list", then add new edge $a$ from $s$ to $s'$ by adding $s'$ into the "$s$.edge_list"; calculate "edge.reward" according to the reward function. Update "edge.value" according to:

$Q(s, a) \coloneqq Q(s, a) + \alpha[R(s, a) + \gamma Q(s', a') - Q(s, a)]$.

# 5 Discussions and evaluation of the model

The RL modeling of the MBF recognition problem passes through the stages of defining all the elements of RL for the MBF domain, as well as the algorithmic implementation of the computational processes of these components. Exact models based on Sperner families and Hansel chains are used, as well as approximate versions of both concepts and their computations are proposed. Hansel models require at least twice the amount of regular memory and reduce the spatial analysis of structures to chain analysis using simple division and the algebra of Hansel chains. The approximations in the calculations associated with Sperner systems turn to inclusion-exclusion type formulas when computing the cardinality of a bundle of intervals, where the sums of the initial sum segments provide lower and upper cardinality estimates in the required precision.

Coarser heuristics are also possible, due to the need to reduce the resulting computational complexity. The simplest heuristics address different kinds of query distances or propagation of points from vertices to current Sperner systems. The whole technique of this block proposes replacing the ideal environment with another, less intelligent and less precise one, with the hope that the necessary computation is simplified and that the RL process converges in a reasonable time frame.

The preliminary experiments conducted involved functions of small dimensionality and mainly focused on clarifying the possibilities of approaches with Sperner and Hansel systems. However, a sufficient volume of experiments has not yet been conducted, and the available volume of trials does not allow to obtain reasonable conclusions to be drawn about the effectiveness of one approach or the other. At this stage, a *Proof of Concept* type result has been established, and an experimental computing environment has been deployed for further work on the basis of the Armenian-French supercomputer

of the Institute for Informatics and Automation Problems of the National Academy of Sciences of Armenia.

# 6 Conclusion and future work

This paper investigated the feasibility of using reinforcement learning to solve combinatorial optimization problems, particularly the query-based reconstruction of monotone Boolean functions.

## Acknowledgments

This paper is partially supported by grant №21T-1B314 of the Science Committee of MESCS RA.

# References

[1] L. Aslanyan, "The discrete isoperimetry problem and related extremal problems for discrete spaces", Problemy kibernetiki, vol.36, pp. 85–128, 1979.

[2] L. Aslanyan and H. Sahakyan, "The splitting technique in monotone recognition", Discrete Applied Mathematics, vol.216, pp. 502–512, 2017.

[3] L. Aslanyan, H. Sahakyan, V. Romanov, G. Da Costa, and R. Kacimi, "Large network target coverage protocols", In 2019 Computer Science and Information Technologies (CSIT), pp. 57–64, IEEE, 2019.

[4] M. Azad, I. Chikalov, S. Hussain, M. Moshkov, B. Zielosko, "Decision Trees with Hypotheses for Recognition of Monotone Boolean Functions and for Sorting", In: Decision Trees with Hypotheses. Synthesis Lectures on Intelligent Technologies. Springer, Cham, 2022. https://doi.org/10.1007/978-3-031-08585-7_6.

[5] R. Bellman, "A Markovian decision process", Journal of Mathematics and Mechanics, vol.6, pp. 679–684, 1957. ISSN 0022-2518.

[6] A. Blum, "Learning a function of r relevant variables (open problem)", In Proceedings of the 16th Annual Conference on Learning Theory and 7th Kernel Workshop, pp. 731-733, 2003.

[7] E. Boros, P. L. Hammer, T. Ibaraki and K. Kawakami, "Identifying 2-monotonic positive Boolean functions in polynomial time, ISA'91, Algorithms, edited by W. L. Hsu and R. C. T. Lee, Springer Lecture Notes in Computer Science, vol. 557, pp. 104-115, 1991.

[8] N. Bshouty and Ch. Tamon, "On the Fourier spectrum of monotone functions". Journal of the ACM (JACM) vol. 43, no. 4, pp. 747-770, 1996.

[9] R. Dedekind, "Über Zerlegungen von Zahlen durch ihre größten gemeinsamen Teiler, Festschrift Hoch", Braunschweig u. ges. Werke, II, pp.103-148, 1897.

[10] P. Frankl and G.OH Katona, "On strengthenings of the intersecting shadow theorem", Journal of Combinatorial Theory, Series A, 184:105510, 2021.

[11] D. N. Gainanov, "On one criterion of the optimality of an algorithm for evaluating monotonic Boolean functions", Computational Mathematics and Mathematical Physics, vol. 24, pp. 176–181, 1984.

[12] M. Gara, T.S. Tasi, P. Balázs, "Machine Learning as a Preprocessing Phase in Discrete Tomography", Lecture Notes in Computer Science, vol. 7346, 2012.

[13] G. Katona, "A theorem of finite sets", In: Theory of Graphs. Academic Press - Akadémiai Kiadó, New York; Budapest, pp. 187-207, 1968.

[14] G. Katona, "A theorem of finite sets", In Classic Papers in Combinatorics, pp. 381–401. Springer, 2009.

[15] G. Hansel, "Sur le nombre des fonctions booléennes monotones de n variables", C. R. Acad. Sci., 262:1088–1090, 1966.

[16] Y. Hu, Y. Yao, W. Lee, "A reinforcement learning approach for optimizing multiple traveling salesman problems over graphs", Knowl-Based Syst 204:106244, 2020.

[17] J.C. Jackson, , H.K.Lee, , R.A. Servedio, , A.Wan, "Learning random monotone DNF", In Approximation, Randomization and Combinatorial Optimization. Algorithms and Techniques. APPROX RANDOM 2008. Lecture Notes in Computer Science, vol 5171. Springer, Berlin, Heidelberg, 2008. https://doi.org/10.1007/978-3-540-85363-3_38

[18] E. B. Khalil, H. Dai, Y. Zhang, B. Dilkina, L. Song. "Learning combinatorial optimization algorithms over graphs", Advances in Neural Information Processing Systems, 30:6351–6361, 2017.

[19] V. K. Korobkov, "On monotone functions of the algebra of logic", Problemy kibernetiki, vol.13, pp. 5–28, 1965.

[20] D. Korshunov, "On the number of monotone boolean functions", Problemy kibernetiki, vol. 38, pp. 5–109, 1981.

[21] J. Kruskal, "The number of simplices in a complex", Mathematical Optimization Techniques, University of California Press, pp. 251–278, 1963.

[22] Zh. Li, Q. Chen, and V. Koltun, "Combinatorial optimization with graph convolutional networks and guided tree search", Advances in neural information processing systems, vol. 31, 2018.

[23] N. Mazyavkina, S. Sviridov, S. Ivanov, and E. Burnaev, "Reinforcement learning for combinatorial optimization: A survey", Computers and Operations Research, vol. 134, 2021.

[24] S. Muroga, "Threshold Logic and Its Applications", Wiley-Interscience, 1971.

[25] R. O'Donnell and R. Servedio, "Learning monotone functions from random examples in polynomial time", Citeseer, 2005.

[26] A. L. C. Ottoni, E. G. Nepomuceno, M.S. d. Oliveira, et al., "Reinforcement learning for the traveling salesman problem with Refueling", Complex & Intelligent Systems, 8, pp.2001–2015, 2022, https://doi.org/10.1007/s40747-021-00444-4.

[27] N. A. Sokolov, “On optimal deciphering of monotone functions of logic algebra”, Journal of Computational Mathematics and Math. Physics, 22(12), pp.1878–1887, 1982.
[28] N. A. Sokolov, “Optimal reconstruction of monotone Boolean functions” Computational Mathematics and Mathematical Physics, 27(6), pp.181–187, 1987.
[29] B.Schröder, “Ordered sets”. Springer 29 (2003): 30.
[30] R. S. Sutton and A.G. Barto, “Reinforcement learning”, The MIT Press, London, 2nd edition, 2018.
[31] G. P. Tonoyan, “Chain partitioning of n-cube vertices and deciphering of monotone Boolean functions”, Computational Mathematics and Mathematical Physics, 19(6), 1979.
[32] L. Valiant, “A theory of the learnable”, Communications of the ACM, 27(11), pp.1134-1142, 1984.
[33] O. Vinyals, M. Fortunato, N. Jaitly, “Pointer networks”, NeurIPS, pp. 2692–2700, 2015.
[34] H. Sahakyan, L. Aslanyan and V. Ryazanov, “On the Hypercube Subset Partitioning Varieties” 2019 Computer Science and Information Technologies (CSIT), Yerevan, Armenia, 2019, pp. 83-88, doi: 10.1109/CSITechnol.2019.8895211.
[35] K.Dohmen, “Improved Bonferroni Inequalities with Applications: Inequalities and Identities of Inclusion-Exclusion Type”. Berlin: Springer-Verlag, 2003.
[36] J. Galambos, J. and I. Simonelli, “Bonferroni-TypeInequalities with Applications”. New York: Springer-Verlag, 1996.