AlphaSeq: Sequence Discovery With Deep Reinforcement Learning

Yulin Shao, *Student Member, IEEE*, Soung Chang Liew, *Fellow, IEEE*, and Taotao Wang, *Member, IEEE*

***Abstract*— Sequences play an important role in many applications and systems. Discovering sequences with desired properties has long been an interesting intellectual pursuit. This article puts forth a new paradigm, AlphaSeq, to discover desired sequences algorithmically using deep reinforcement learning (DRL) techniques. AlphaSeq treats the sequence discovery problem as an episodic symbol-filling game, in which a player fills symbols in the vacant positions of a sequence set sequentially during an episode of the game. Each episode ends with a completely filled sequence set, upon which a reward is given based on the desirability of the sequence set. AlphaSeq models the game as a Markov decision process (MDP) and adapts the DRL framework of AlphaGo to solve the MDP. Sequences discovered improve progressively as AlphaSeq, starting as a novice, and learns to become an expert game player through many episodes of game playing. Compared with traditional sequence construction by mathematical tools, AlphaSeq is particularly suitable for problems with complex objectives intractable to mathematical analysis. We demonstrate the searching capabilities of AlphaSeq in two applications: 1) AlphaSeq successfully rediscovers a set of ideal complementary codes that can zero-force all potential interferences in multi-carrier code-division multiple access (CDMA) systems and 2) AlphaSeq discovers new sequences that triple the signal-to-interference ratio—benchmarked against the well-known Legendre sequence—of a mismatched filter (MMF) estimator in pulse compression radar systems.**

***Index Terms*— AlphaGo, deep reinforcement learning (DRL), Monte Carlo tree search (MCTS), multi-carrier code-division multiple access (MC-CDMA), pulse compression radar.**

I. Introduction

A SEQUENCE is a list of elements arranged in a certain order. Prime numbers arranged in ascending order, for example, are a sequence [1]. The arrangements of nucleic acids in DNA polynucleotide chains are also sequences [2].

Discovering sequences with desired properties is an intellectual pursuit with important applications [1]. In particular, sequences are critical components in many information systems. For example, cellular code-division multiple access (CDMA) systems make use of spread spectrum sequences to distinguish signals from different users [3]; pulse compression radar systems make use of probe pulses modulated by phase-coded sequences [4] to enable high-resolution detection of objects at a large distance.

Sequences in information systems are commonly designed by algebraists and information theorists using mathematical tools such as finite field theory, algebraic number theory, and character theory. However, the design criterion for a good sequence may be complex and cannot be put into a clean mathematical expression for a solution by the available mathematical tools. Faced with this problem, sequence designers may do the following two things.

1) Overlook the practical criterion and simplify the requirements to make the problems analytically tractable. In so doing, a disconnect between reality and theory may be created.
2) Introduce additional but artificial constraints absent in the original practical problem. In this case, the analytical solution is only valid for a subset of sequences of interest. For example, the protocol sequences in [5] are constructed by means of the Chinese remainder theorem (CRT); hence, the number of supported users is restricted to a prime number.

Yet, a third approach is to find the desired sequences algorithmically. This approach rids us of the confines imposed by analytical mathematical tools. On the other hand, the issue becomes whether good sequences can be found within a reasonable time by algorithms. Certainly, to the extent that desired sequences can be found by a random search algorithm within a reasonable time, then the problem is solved. Most desired sequences, however, cannot be found so easily, and algorithms with complexity polynomial in the length of the sequences are not available.

Reinforcement learning (RL) is an important branch of machine learning [6] known for its ability to derive solutions for Markov decision processes (MDPs) [7] through a learning process. A salient feature of RL is "learning from interactions." Fig. 1 illustrates a framework of RL. In the framework, an agent interacts with an environment in a sequence of discrete-time steps. At time step t, the agent observes that the environment is in state s_t. Based on the observation of s_t, the agent then takes an action a_t, which results in the agent receiving a reward $R(s_{t+1})$ and the environment moving to

Manuscript received October 3, 2018; revised January 14, 2019, May 1, 2019, and August 7, 2019; accepted September 16, 2019. Date of publication October 21, 2019; date of current version September 1, 2020. This work was supported in part by the General Research Funds established under the University Grant Committee of the Hong Kong Special Administrative Region, China, under Project 14200417. *(Corresponding author: Soung Chang Liew.)*

Y. Shao and S. C. Liew are with the Department of Information Engineering, The Chinese University of Hong Kong, Hong Kong (e-mail: sy016@ie.cuhk.edu.hk; soung@ie.cuhk.edu.hk).

T. Wang was with the Department of Information Engineering, The Chinese University of Hong Kong, Hong Kong. He is now with the College of Information Engineering, Shenzhen University, Shenzhen 518061, China (e-mail: ttwang@szu.edu.cn).

Color versions of one or more of the figures in this article are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TNNLS.2019.2942951

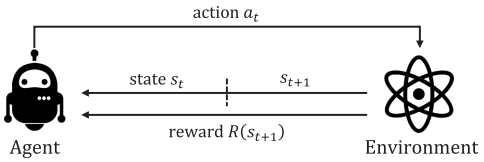


Fig. 1. Agent-environment interactions in RL. Given the observation that the environment is in state s_t, the agent follows its current policy and takes action a_t. The environment then moves to state s_{t+1} and feedbacks a reward $R(s_{t+1})$ [6].

state s_{t+1}. RL framework can also be episodic, in which the agent-environment interactions are broken into sessions called episodes. The environment will feedback a reward $R(s_T)$ only at a terminal state, i.e., at the end of one episode. The mapping from s_t to a_t is referred to as a policy function. The aim of the policy is to maximize the expected reward received at the end of the episode. This policy function could be deterministic, in which case a specific action a_t is always taken upon a given state s_t. The policy could also be probabilistic, in which case the action taken upon a given state is described by a conditional probability $P(a_t|s_t)$. The objective of the agent is to learn an expected reward maximizing policy after going through multiple episodes.[1] The agent may begin with bad policies early on, but as it gathers experiences from successive episodes, the policy gets better and better.

The latest trend in RL research is to integrate the recent advances of deep learning [8] into the RL framework [9]–[11]. RL that makes use of deep neural networks (DNNs) to approximate the optimal policy function—-directly or indirectly—is referred to as deep RL (DRL). DRL allows RL algorithms to be applied when the number of possible state-action pairs is enormous and that traditional function approximators cannot approximate the policy function accurately. The recent success of DRL in game playing, natural language processing, and autonomous vehicle steering (see the excellent survey in [11]) has demonstrated its power in solving complex problems that thwart conventional approaches.

This article puts forth a DRL-based paradigm, referred to as AlphaSeq, to discover a set of sequences with desired properties algorithmically. The essence of AlphaSeq is as follows.

1) AlphaSeq treats sequence-set discovery—a sequence set consists of one or more sequences—as an episodic symbol-filling game. In each episode of the game, AlphaSeq fills symbols into vacant sequence positions in a consecutive manner until the sequence set is completely filled, whereupon a reward with a value between -1 and 1 is returned. The reward is a nonlinear function of a metric that quantifies the desirability of the sequence set. AlphaSeq aims to maximize the reward. It learns to do by playing many episodes of the game, improving itself along the way.
2) AlphaSeq treats each intermediate state, with some sequence positions filled and others vacant in the game, as an image. Each position is a pixel of the image. Given an input state (image), AlphaSeq makes use of a DNN to recognize it and approximate the optimal policy that maximizes the reward. The DRL framework in AlphaSeq is adapted from AlphaGo [12], in which DNN-guided Monte Carlo Tree Search MCTS) is used to select each move in the game. As in AlphaGo, there is an iterative self-learning process in AlphaSeq in that the experiences from the DNN-guided MCTS game playing are used to train the DNN; and the trained DNN, in turn, improves future game playing by the DNN-guided MCTS.
3) We introduce two techniques in AlphaSeq that are absent in AlphaGo for our applications to search sequences. The first technique is to allow AlphaSeq to make ℓ moves at a time (i.e., filling ℓ sequence positions at a time). Obviously, this technique is not applicable to the game of Go, hence AlphaGo. The choice of ℓ is a complexity tradeoff between the MCTS and the DNN. The second technique, dubbed "segmented induction," is to change the reward function progressively to guide AlphaSeq toward good sequences in its learning process. In essence, we set a low target for AlphaSeq initially so that many sequence sets can have rewards close to 1, with few having rewards close to -1. As AlphaSeq plays more and more episodes of the game, we progressively raise the target so that fewer and fewer sequence sets have rewards close to 1, with more having rewards close to -1. In other words, the game becomes more and more demanding as AlphaSeq, starting as a novice, and learns to become an expert player.

We demonstrate the capability of AlphaSeq to discover two types of sequences. First, we use AlphaSeq to rediscover a set of complementary codes for multi-carrier (MC)-CDMA systems. In this application, AlphaSeq aims to discover a sequence set for which potential interferences in the MC-CDMA system can be canceled by simple signal processing. This particular problem already has analytical solutions. Our goal, here, is to test if AlphaSeq can rediscover these analytical solutions algorithmically rather than analytically. Second, we use AlphaSeq to discover new phase-coded sequences superior to the known sequences for pulse compression radar systems. Specifically, our goal is to find phase-coded sequences commensurate with the mismatched filter (MMF) estimator so that the estimator can yield an output with a high signal-to-interference ratio (SIR). The optimal sequences for MMF are not known and there is currently no known sequence that is provably optimal when the sequence is large. Benchmarked against the Legendre sequence [13], the sequence discovered by AlphaSeq triples the SIR, achieving 5.23-dB mean square error (MSE) gains for the estimation of radar cross sections in pulse compression radar systems.

The remainder of this article is organized as follows. Section II formulates the sequence discovery problem and

[1]RL shares the same mathematical principle as that of dynamic programming (DP). To learn the optimal policy, RL algorithms typically contain two interacting processes: policy evaluation and policy improvement. We refer the reader to the exposition in [6], in which Section IV.1 explains how policy evaluation predicts the state-value function for an arbitrary policy, and Section IV.2 explains how policy improvement improves the policy with respect to the current state-value function. Overall, these two processes interact with each other as a generalized policy iteration (Section IV.6), enabling the convergence to the optimal value function and an optimal policy.

outlines the DRL framework of AlphaSeq. Sections III and IV present the applications of AlphaSeq in MC-CDMA systems and pulse compression radar systems, respectively. Section V concludes this article. Throughout this article, lowercase bold letters denote vectors and uppercase bold letters denote matrices.

II. Methodology

A. Problem Formulation

We consider the problem of discovering a sequence set $\mathcal{C}$, the desirability of which is quantified by a metric $\mathcal{M}(\mathcal{C})$. Set $\mathcal{C}$ consists of K different sequences of the same length N, i.e., $\{\boldsymbol{c}_k : k = 0, 1, \ldots, K-1\}$, where the kth sequence is given by $\boldsymbol{c}_k = (c_k[0], c_k[1], \ldots, c_k[N-1])$. Each symbol of the sequences in $\mathcal{C}$ (i.e., $c_k[n]$) is drawn from a discrete set $\mathcal{A}$. Without loss of generality, this article focuses on binary sequences. That is, $\mathcal{A}$ is two-valued, and we can simply denote these two values by 1 and -1. The metric function $\mathcal{M}(\mathcal{C})$ varies with application scenarios. It is generally a function of all K sequences in $\mathcal{C}$. The optimal metric value $\mathcal{M}^*$ (i.e., the desired metric value) is achieved when $\mathcal{C} = \mathcal{C}^*$. Our objective is to find an optimal sequence set $\mathcal{C}^*$ that yields $\mathcal{M}^*$. For binary sequences, the complexity of the exhaustive search for $\mathcal{C}^*$ is $\mathcal{O}(2^{NK})$, which is prohibitive for large N and K.

This sequence discovery problem can be transformed into an MDP. Specifically, we treat sequence-set discovery as a symbol-filling game. One play of the game is one episode, and each episode contains a series of time steps. In each episode, the player (agent) starts from an all-zero state (i.e., all the symbols in the set are 0) and takes one action per time step based on its current action policy. In each time step, ℓ symbols in the sequence set are assigned with the value of 1 or -1, replacing the original 0 value. We emphasize that the player can only determine the values of the ℓ symbols but not their positions. The ℓ positions are predetermined: a simple rule is to place symbols sequence by sequence (specifically, we first place symbols in one sequence). When this sequence is completed-filled, we turn to fill the next sequence, and so on and so forth. This rule will be used throughout this article unless specified otherwise). An episode ends at a terminal state after $\lceil NK/\ell \rceil$ time steps, whereupon a complete set $\mathcal{C}$ is obtained. In the terminal state, we measure the goodness of $\mathcal{C}$ by $\mathcal{M}(\mathcal{C})$ and return a reward $\mathcal{R}(\mathcal{C})$ for this episode to the player, where $\mathcal{R}(\mathcal{C})$ is, in general, a nonlinear function of $\mathcal{M}(\mathcal{C})$. For this MDP, episodic RL can be used to learn a policy that makes sequential decisions to maximize the reward $\mathcal{R}(\mathcal{C})$. In a general RL setup, the agent's objective at each state s_t is to maximize the accumulated future reward $\sum_i \gamma^{i-1} R_{t+i}$, where γ is a discount factor. The MDP above, by contrast, restricts the reward to only terminal state since the quality of the sequence set cannot be evaluated until all the elements are fixed. This is a delayed reward setup, the goal at each state is now to maximize the end-of-episode reward.

B. Methodology

Given the MDP, a tree can be constructed by all possible states in the game. In particular, the root vertex is the all-zero state, and each vertex of the tree corresponds to a possible state, i.e., a partially filled sequence-set pattern (completely filled at a terminal state). The depth of the tree equals the number of time steps in an episode (i.e., $\lceil NK/\ell \rceil$), and each vertex has exactly 2^ℓ branches. In each episode, the player will start from the root vertex and make sequential decisions along the tree based on its current policy until reaching a leaf vertex, whereupon a reward will be obtained. Given any vertex v_i and an action, the next vertex v_{i+1} is conditionally independent of all previous vertices and actions, i.e., the transitions on the tree satisfy the Markov property.

The objective of the player is then to reach a leaf vertex with the maximum reward. Toward this objective, the player performs the following.

1) Distinguishing good states from bad states—A reward is given to the player only upon its reaching a terminal stage. Although traversing the intermediate stage, the player must distinguish good intermediate states from bad intermediate states so that it can navigate toward a good terminal stage. In particular, the player must learn to approximate the expected end rewards of intermediate states: this is, in fact, a process of value function approximation (in RL, the value of a state refers to the expected reward of being in that state, and a value function is a mapping from states to values. For terminal states, the value function is exactly the reward function). Moreover, we can imagine each state to be an image with each symbol being a pixel and make use of a DNN to approximate the expected rewards of the "images."
2) Improving action policy based on cognition of subsequent states. Starting as a tabula rasa, the player's initial policy in earlier episodes is rather random. To gradually improve the action policy, the player can leverage the instrument of MCTS. MCTS is a simulated look-ahead tree search. At a vertex, MCTS can estimate the prospects of subsequent vertices by simulating multiple actions along the tree. The information collected during the simulations can then be used to decide the real action to be taken at this vertex.[2]

A successful combination of DNN and MCTS has been demonstrated in AlphaGo [10], [12], [16], where the authors use DNN to assess the vertices during the MCTS simulation, as opposed to using random rollouts in standard MCTS [14]. In this article, we adapt the DRL framework in AlphaGo[3] to solve the sequence set discovery problem associated with the underlying MDP. In deference to AlphaGo, we refer to this sequence discovering framework as "AlphaSeq."

The overall algorithmic framework of AlphaGo/AlphaSeq can be outlined as an iterative "game-play with MCTS" and "DNN-update" process, as shown in Fig. 2. On the one hand,

[2]The main concept in MCTS is tree policy. It determines how we sample the tree and select nodes. For a general overview on the core algorithms and variations, we refer the reader to the excellent survey [14] (in particular, the most popular algorithm in the MCTS family, the Upper Confidence Bound for Trees (UCT), is introduced in Section III.3 of [14]). Reference [15] provides a more rigorous proof of the optimality of UCT. The authors showed that the probability that the UCT selects the optimal action converges to 1 at a polynomial rate.

[3]AlphaGo itself is evolving, the DRL framework in this article is based on AlphaGo Zero [12] and AlphaZero [16].

Fig. 2. Iterative algorithmic framework of AlphaGo/AlphaSeq. Improved DNN promotes the MCTS so that "game-play" generates experiences with higher quality; higher quality experiences can further enhance the DNN.

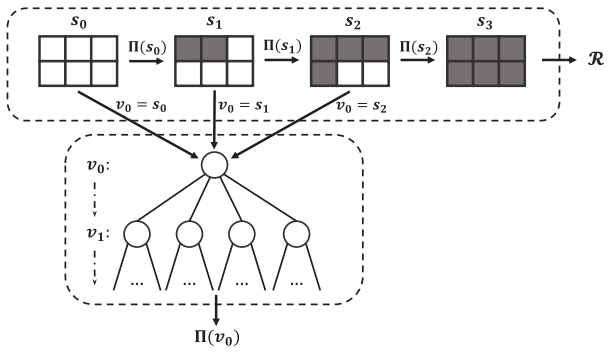


Fig. 3. Episode of game, where $K = 2$, $N = 3$, and $\ell = 2$. The NK positions are represented by colored squares: gray means that the positions are filled, while white means that the positions are vacant. At each time step, following the MCTS output $\mathbf{\Pi}$, the player fills ℓ positions with value 1 or -1.

"game-play with MCTS" provides experiences to train the DNN so that the DNN can improve its assessments of the goodness of the states in the game. On the other hand, better evaluation on the states by the DNN allows the MCTS to make better decisions, which, in turn, provide higher quality experiences to train the DNN. Through an iterative process, the MCTS and the DNN mutually enhance each other in a progressive manner over an underlying RL process.

In what follows, we dissect these two components and describe the relationship between them with more details. Differences between AlphaSeq and AlphaGo are presented at the end of this section. Further implementation details can be found in Appendix A.

1) Input and Output of DNN: The DNN is designed to estimate the value function and policy function of an intermediate state. The value function is the estimated expected terminal reward given the intermediate state. Specifically, the output of DNN can be expressed as $(\boldsymbol{P}, \mathcal{R}') = \psi_\theta(s_i)$: each time we feed an intermediate state s_i into the DNN ψ with coefficients θ, it will output a reward estimation $\mathcal{R}'$ (value function estimation) and a probabilistic move-selection policy $\boldsymbol{P}$ (policy function estimation, policy $\boldsymbol{P}$ is a distribution over all possible next moves given the current state s_i).

2) Game-Play With MCTS: The first part of the algorithm iteration in Fig. 2 is game-play with MCTS. As illustrated in Fig. 3, we play the game under the guidance of MCTS. The upper half of Fig. 3 presents all the states in an episode, where squares represent the positions in the sequence set: gray squares mean that the position has already been filled (with value 1 or -1); white squares mean that the position is still vacant (with value 0). The initial state of each episode is an all-zero state s_0. In state s_i, the player will follow a probabilistic policy $\mathbf{\Pi}(s_i)$ (a distribution over all 2^ℓ possible moves given by MCTS, not the raw policy estimation $\boldsymbol{P}$ of DNN) to choose ℓ symbols to fill in the next ℓ positions in the sequence set. This action yields a new state s_{i+1}.

The bottom half of Fig. 3 shows the MCTS process at each state s_i, where each circle (vertex) represents a possible state in the look-ahead search. In the MCTS for state s_i, we first set the root vertex v_0 to be s_i and initialize a "visited tree" (this visited tree is used to record all the vertices visited in the MCTS. It is initialized to have only one root vertex). Look-ahead simulations are then performed along the visited tree starting at the root vertex. Each simulation traces out a path of the visited tree and terminates when an unseen vertex v_L is encountered. This unseen vertex will then be evaluated by DNN and added to the visited tree (i.e., a newly added vertex v_L will be given the metric as $\psi_\theta(v_L) = (\boldsymbol{P}_L, \mathcal{R}'_L)$ to aid future simulations in evaluating which next move to select if the same vertex v_L is visited again). As more and more simulations are performed, the tree grows in size. The metric used in selecting next move for the vertices will also change [i.e., (20) and (21) in Appendix A] as the vertices are visited more and more in successive simulations. In a nutshell, estimated good vertices are visited frequently, while estimated bad vertices are visited rarely. The resulting move-selection distribution at state s_i, i.e., $\mathbf{\Pi}(s_i) = (\pi_0, \pi_1, \ldots, \pi_{2^\ell - 1})$, is generated from the visiting counts of the root vertex's children in MCTS at states s_i.

Back to the upper part of Fig. 3, after $\lceil NK/\ell \rceil$ time steps, the player obtains a complete sequence set $\mathcal{C}$ with metric value $\mathcal{M}(\mathcal{C})$ that gives a reward $\mathcal{R}(\mathcal{C})$. Then, we feed the $\mathcal{R}(\mathcal{C})$ to each state s_i in this episode and store $(s_i, \mathbf{\Pi}(s_i), \mathcal{R})$ as an experience. One episode of game-play gives us $\lceil NK/\ell \rceil$ experiences.

3) DNN Update: The second part of the algorithm iteration in Fig. 2 is the training of the DNN based on the accumulated experiences over successive episodes. First, from the description above, we know that MCTS is guided by DNN. The capability of DNN determines the performance of MCTS since a better DNN yields more accurate evaluation of the vertices in MCTS. In the extreme, if the DNN perfectly knows which sequence-set patterns are good and which are bad, then the MCTS will always head toward an optimal direction, hence the chosen moves are also optimal. However, the fact is, DNN is randomly initialized, and its evaluation on vertices are quiet random and inaccurate initially. Thus, our goal is to improve this DNN using the experiences generated from game-play with MCTS.

In the process of DNN update, the DNN is updated by learning the latest experiences accumulated in the game-play. Given experience $(s_i, \mathbf{\Pi}(s_i), \mathcal{R})$ and $\psi_\theta(s_i) = (\boldsymbol{P}, \mathcal{R}')$, 1) the real reward $\mathcal{R}$ can be used to improve the value-function approximation $\mathcal{R}'$ of DNN and 2) the policy $\mathbf{\Pi}(s_i)$ given by MCTS at state s_i can be used to improve the policy estimation $\boldsymbol{P}(s_i)$ of DNN (policy $\mathbf{\Pi}(s_i)$ is generally more powerful than the raw output $\boldsymbol{P}(s_i)$ of DNN). Thus, the training process is to make $\boldsymbol{P}$ and $\mathcal{R}'$ more closely match $\mathbf{\Pi}$ and $\mathcal{R}$.

Remark: When we play games with MCTS to generate experiences, the Dirichlet noise is added to the prior

probability of root node v_0 to induce exploration, as that in AlphaGo [12]. These games are also called noisy games. Instead of noisy games, we can also play noiseless games in which the Dirichlet noise is removed. Following the practice of AlphaGo, we play noisy games to generate the training experiences, but play noiseless games to evaluate the performance of AlphaSeq whose MCTS is guided by a particular trained DNN.

Overall, in one iteration, we: 1) play G episodes of noisy games with ψ_θ-guided MCTS to generate experiences, where ψ_θ is the current DNN; 2) use experiences gathered in the latest $z \times G$ episodes of games to train for a new DNN $\psi_{\theta'}$; and 3) assess the new DNN $\psi_{\theta'}$ by running 50 noiseless games with $\psi_{\theta'}$-guided MCTS.

In the next iteration, we generate further experiences by playing G episodes of noisy games with $\psi_{\theta'}$-guided MCTS. Then, these experiences are further used to train for yet another new DNN and so on and so forth. The pseudocode for AlphaSeq is given in Algorithm 1.

Algorithm 1: AlphaSeq

Initialization:
Initialize parameters z and DNN update cycle G.
Initialize a DNN ψ_θ with parameter θ.
Set episode $g = 0$.
while 1 **do**
  Self-play to gain experience:
  Play one episode of game, output each immediate state s_i, the corresponding $\Pi(s_i)$ given by MCTS, and the discovered sequence set $\mathcal{C}$ when this episode ends.
  Compute metric $\mathcal{M}(\mathcal{C})$ and reward $\mathcal{R}(\mathcal{C})$.
  $\forall s_i$, store $(s_i, \Pi(s_i), \mathcal{R}(\mathcal{C}))$ as experience.
  $g = g + 1$.
  DNN update:
  If $\mathrm{mod}(i, G) == 0$ **then**
    Train DNN using the experiences accumulated in the latest $z \times G$ episodes, get new parameters θ'.
    Assess current AlphaSeq with new parameters $\psi_{\theta'}$ by playing 50 noiseless games.
    $\psi_\theta = \psi_{\theta'}$.
  end
end

In the following, we highlight some differences between AlphaSeq and AlphaGo.

1) In AlphaGo, the total number of legal states is $\mathcal{O}(3^{NK})$ (in Go, $N = K = 19$; each position can be occupied by no stones, a white stone, or a black stone). If we allow AlphaSeq to fill symbol positions in arbitrary order, then the complexity would be the same as AlphaGo in terms of the parameters N and K. However, for AlphaSeq, we impose the order in which symbol positions are filled to reduce complexity. Now, the number of legal states reduces to

$$\sum_{t=0}^{\lceil NK/\ell \rceil} 2^{t\ell} = \frac{2^{NK+\ell} - 1}{2^\ell - 1} \tag{1}$$

that is, the state at the beginning of the time step t has 2^{tl} possible values. We found that imposing this restriction, while reducing complexity substantially, does not compromise the optimality of the sequence found.

2) In AlphaSeq, the choice of ℓ is a complexity tradeoff between MCTS and DNN; in AlphaGo, ℓ is always 1. As mentioned above, the universe of all states in the game forms a tree. The depth of the tree is $\lceil NK/\ell \rceil$, which is the number of steps in Fig. 3 from left to right. This is exactly the number of MCTS we need to run in an episode. Thus, the larger the ℓ, the fewer the MCTS we need to run. On the other hand, large ℓ yields more legal moves (i.e., 2^ℓ) in each state, hence burdening the DNN with a larger action space. Overall, given N and K, for small ℓ, for example, $\ell = 1$, the mission of DNN is light since it only needs to determine to place 1 or -1 in the next position. However, the number of MCTS we need to run in an episode is up to NK. In contrast, for large ℓ, for example, $\ell = K$, the number of MCTS we need to run in an episode is reduced to N, but the DNN is burdened with a heavier task because it needs to evaluate 2^K possible moves for each state.

3) In the game of Go, the board is invariant to rotation and reflection. Thus, we should augment the training data to let DNN learn these features. Specifically, in AlphaGo Zero, each experience (board state and move distribution) can be transformed by rotation and reflection to obtain extra training data, and the state in an experience is randomly transformed before the experience is fed to the DNN [12]. On the other hand, in our game, no rotation or reflection is required because all positions are predetermined. Any rotated or reflected state is an illegal state.

In the following sections, we demonstrate the searching capabilities of AlphaSeq in two applications: in Section III, we use AlphaSeq to rediscover an ideal complementary code set for MC-CDMA systems; and in Section IV, we use AlphaSeq to discover a new phase-coded sequence for pulse compression radar systems.

III. Rediscover Ideal Complementary Code for Multi-Carrier CDMA

CDMA is a multiple-access technique that enables numerous users to communicate in the same frequency band simultaneously [3]. The fundamental principle of CDMA communications is to distinguish different users (or channels) by unique codes preassigned to them. Thus, CDMA code design lies at the heart of CDMA technology.

A. Codes in Legacy CDMA Systems

Existing cellular CDMA systems work on a one-code-per-user basis [3], [17]. For example, the code set is designed such that exactly one code is assigned to each user, e.g., the orthogonal variable spreading factor (OVSF) code set used in W-CDMA downlink, the m-sequence set used in CDMA2000 uplink, and the Gold sequence set used in W-CDMA uplink [18]. However, legacy CDMA systems are self-jamming systems since the code sets being used

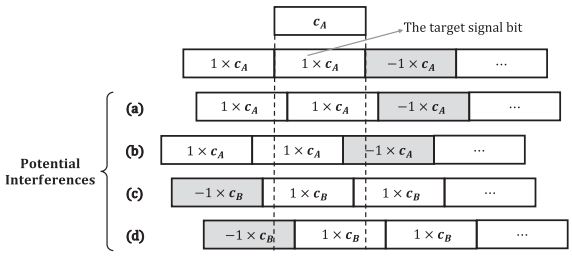


Fig. 4. Interferences caused by user asynchronies (misalignments of bit boundaries), multi-paths, and random signs of consecutive bits, in CDMA uplink. To decode user A's data, the receiver correlates the received signal with code $\boldsymbol{c_A}$. Interferences are induced by (a) cyclic autocorrelation of $\boldsymbol{c_A}$, (b) flipped autocorrelation of $\boldsymbol{c_A}$, (c) cyclic cross correlation between $\boldsymbol{c_A}$ and $\boldsymbol{c_B}$, and (d) flipped cross correlation between $\boldsymbol{c_A}$ and $\boldsymbol{c_B}$.

cannot guarantee user orthogonality under practical constraints and considerations, such as user asynchronies, multipath effects, and random signs of consecutive bits[4] of user data streams [19].

In CDMA uplink, each user spreads its signal bits by modulating the assigned code, and the signals from multiple users overlap at the receiver. To decode a user A's signal bit, as shown in Fig. 4, the receiver cross correlates the received signal with the locally generated code of user A. However, due to user asynchronies, multi-paths, and random signs in consecutive bits, the correlation results can suffer from interferences introduced by multiple paths of user A's signal or signal from another user B. The potential interferences can be computed by the correlations between the signal bit and two overlapping interfering bits: when the signs of the two interfering bits are the same, the interferences are cyclic correlation functions [i.e., Fig. 4(a) and (c)]; when the signs of the two interfering bits are different, the interferences are flipped correlation functions [i.e., Fig. 4(b) and (d)]. On the other hand, CDMA downlink is a synchronous CDMA system and there are no asynchronies among signals of different users. However, multi-path and random signs in consecutive bits can still cause interferences through the above correlations among codes.

Mathematically, it has been proven that the ideal one-code-per-user code set that simultaneously zero forces the above correlation functions does not exist [20]. Code sets used in legacy CDMA systems tradeoff among these correlation functions. For example, the m-sequence set has nearly ideal cyclic autocorrelation property (to be exact, the autocorrelation function of the m-sequence is -1 for any nonzero shift, hence is "nearly" optimal), while its cyclic cross correlation function (CCF) and flipped correlation function are unbounded. The Gold sequence set and the Kasami sequence set (candidate in W-CDMA) have better cyclic cross correlation properties and acceptable cyclic autocorrelation properties, but their flipped correlations are unbounded [18].

[4]In CDMA, "bit" refers to the baseband modulated information symbols (only BPSK/QPSK modulated symbols are considered in this article, in general, it can be shown that the codes discussed in this section are applicable for higher order modulations), while "chip" refers to the entries in the spread spectrum code. Thus, with respect to the nomenclature in Section II, "chips" in CDMA corresponds to "symbol" of a code sequence in Section II.

B. Multi-Carrier CDMA and Ideal Complementary Codes

The limitations of legacy CDMA systems motivate researchers to develop MC-CDMA (MC-CDMA) systems where complementary codes can be used to simultaneously null all correlation functions among codes that may cause interferences [19].

The basic idea of complementary codes is to assign a flock of M element codes to each user, as opposed to just one code in legacy CDMA systems. In MC-CDMA uplink [17], the signal bits of a user are spread by each of its M element codes and sent over M different subcarriers. When passing through the channel, the M subcarriers can be viewed as M separate virtual channels that have the same delay. The receiver first despreads the received signal in each individual subcarrier (i.e., correlate the received signal in each subcarrier with the corresponding element code) and sums up the despreading outcomes of all M subcarriers. In other words, the operations in each individual channel are the same as legacy CDMA systems: the new step is the summing of the outputs of the M virtual channels, which cancels out the interferences induced by individual correlations in the underlying subcarriers.

To be specific, let us consider an MC-CDMA system with J users, where a flock of M element codes of length N is assigned to each user. An ideal complementary code set $\mathcal{C} = \{c_j^m[n] : j = 0, 1, \ldots, J-1; m = 0, 1, \ldots, M-1; n = 0, 1, \ldots, N-1\}$ that can enable interference-free MC-CDMA systems is a code set that meets the following criteria simultaneously.

1) *Ideal Cyclic Autocorrelation Function (CAF):* For the M element codes assigned to a user j, i.e., $\{\boldsymbol{c}_j^m : m = 0, 1, \ldots, M-1\}$, the sum of the CAF of each code is zero for any nonzero shift

$$\mathrm{CAF}_j[\upsilon] = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} c_j^m[n] c_j^m[n+\upsilon] = 0 \qquad (2)$$

where delay (chip-level) $\upsilon = 1, 2, .., N-1$. Hereinafter, the index additions in the square brackets refer to modulo-N additions.

2) *Ideal CCF:* For two flocks of codes assigned to users j_1 and j_2, i.e., $\{\boldsymbol{c}_{j_1}^m, \boldsymbol{c}_{j_2}^m : m = 0, 1, \ldots, M-1\}$, the sum of their CCFs is always zero irrespective of the relative shift

$$\mathrm{CCF}_{j_1, j_2}[\upsilon] = \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} c_{j_1}^m[n] c_{j_2}^m[n+\upsilon] = 0 \qquad (3)$$

where delay $\upsilon = 0, 1, 2, .., N-1$ and $j_1 \neq j_2$.

3) *Ideal Flipped Correlation Function (Fcf):* For two flocks of codes assigned to users j_1 and j_2, i.e., $\{\boldsymbol{c}_{j_1}^m, \boldsymbol{c}_{j_2}^m : m = 0, 1, \ldots, M-1\}$, the sum of their flipped correlation functions is always zero for any nonzero shift (flipped correlation is only defined for nonzero delay)

$$\mathrm{FCF}_{j_1, j_2}[\upsilon] = \sum_{m=0}^{M-1} \left\{ \sum_{n=0}^{N-\upsilon-1} c_{j_1}^m[n] c_{j_2}^m[n+\upsilon] - \sum_{n=N-\upsilon}^{N-1} c_{j_1}^m[n] c_{j_2}^m[n+\upsilon] \right\} = 0 \qquad (4)$$

where delay $\upsilon = 1, 2, .., N-1$; j_1 and j_2 can be the same (flipped autocorrelation function) or different (flipped cross correlation function).

Some known mathematical constructions of ideal complementary codes are available in [17]. In this section, we make use of AlphaSeq to rediscover a set of ideal complementary codes. Our aim is to investigate and evaluate the searching capability of AlphaSeq, i.e., whether it can rediscover an ideal complementary code set and how it goes about doing so. Furthermore, we would like to investigate the impact of the hyperparameters used in the search algorithm on the overall performance of AlphaSeq, so as to obtain useful insights for discovering other unknown sequences (e.g., in Section IV, we will make use of AlphaSeq to discover phase-coded sequences for pulse compression radar systems)

C. AlphaSeq for MC-CDMA

In this section, we use AlphaSeq to rediscover an ideal complementary code set for MC-CDMA systems. As stated above, the ideal complementary code set is the code set that fulfills the three criteria in (2)–(4). In this context, given a sequence set $\mathcal{C}$, we define the following metric function to measure how good set $\mathcal{C}$ is for MC-CDMA systems.

1) Metric Function: For a sequence set $\mathcal{C} = \{c_j^m[n] : j = 0, 1, \ldots, J-1; m = 0, 1, \ldots, M-1; n = 0, 1, \ldots, N-1\}$ consisting of MJ sequences of the same length N, the metric function $\mathcal{M}(\mathcal{C})$ in the following reflects how good $\mathcal{C}$ is for MC-CDMA systems:

$$\mathcal{M}(\mathcal{C}) = \sum_{j=0}^{J-1}\sum_{\upsilon=1}^{N-1}|\mathrm{CAF}_j[\upsilon]| + \sum_{j_1=0}^{J-1}\sum_{j_2=j_1+1}^{J-1}\sum_{\upsilon=0}^{N-1}|\mathrm{CCF}_{j_1,j_2}[\upsilon]| + \sum_{j_1=0}^{J-1}\sum_{j_2=j_1}^{J-1}\sum_{\upsilon=1}^{N-1}|\mathrm{FCF}_{j_1,j_2}[\upsilon]|. \quad (5)$$

Note that our desired metric value $\mathcal{M}^* = \inf \mathcal{M}(\mathcal{C}) = 0$. For AlphaSeq, the objective is then to discover the sequence set that minimizes this metric function.

As an essential part of the training paradigm in AlphaSeq, a reward function is needed to map a found sequence set $\mathcal{C}$ to a reward $\mathcal{R}(\mathcal{C})$. In general, we could design this reward function to be a linear (or nonlinear) mapping from the value range of the metric function to the interval $[-1, 1]$. This is, in fact, a normalization process to fit general objectives to the architecture of AlphaSeq (specifically, normalizing the rewards of different problems allows these problems to share the same underlying hyperparameters in DNN and MCTS of the AlphaSeq architecture). To rediscover the ideal complementary code, we define the reward function as follows.

2) Reward Function: For any sequence set $\mathcal{C}$ with metric $\mathcal{M}(\mathcal{C})$, the reward $\mathcal{R}(\mathcal{C})$ for MC-CDMA systems is defined as

$$\mathcal{R}(\mathcal{C}) = \begin{cases} 1 - \dfrac{2\mathcal{M}(\mathcal{C})}{\mathcal{M}_u}, & \text{If } 0 \le \mathcal{M}(\mathcal{C}) \le \mathcal{M}_u \\ -1, & \text{If } \mathcal{M}(\mathcal{C}) > \mathcal{M}_u \end{cases} \quad (6)$$

where $[0, \mathcal{M}_u]$ is the search range of $\mathcal{M}(\mathcal{C})$: when $\mathcal{M}(\mathcal{C}) = \mathcal{M}_u$, then $\mathcal{R}(\mathcal{C}) = -1$; and when $\mathcal{M}(\mathcal{C}) = 0$, then $\mathcal{R}(\mathcal{C}) = 1$. We initially set $\mathcal{M}_u = \max_{\mathcal{C}} \mathcal{M}(\mathcal{C})$ [see Appendix B for the derivation of $\max_{\mathcal{C}} \mathcal{M}(\mathcal{C})$] and initialize the DNN to ψ_{θ_0} (i.e., the parameters in the DNN are randomly set to θ_0) to play 50 noiseless games. Then, $\mathcal{M}_u$ is set as the mean metric of the 50 sequences found by these 50 noiseless games, i.e., $\mathcal{M}_u = E[\mathcal{M}]$. After this, $\mathcal{M}_u$ will not be changed anymore in future games. We specify that the initial games do not find good sequences, but, nevertheless, the 50 sequences yield an $E[\mathcal{M}]$ much lower than $\max_{\mathcal{C}} \mathcal{M}(\mathcal{C})$. Using $E[\mathcal{M}]$ as $\mathcal{M}_u$ increases the slope of the first line in (6).

Based on the metric function and reward function defined above, we implemented AlphaSeq and trained DNN to rediscover an ideal complementary code for MC-CDMA. A known ideal complementary code [17] is chosen as benchmark.

3) Benchmark: When $J = 2$, $M = 2$, and $N = 8$, the ideal complementary code set exists. The mathematical constructions in [17] gives us

$$\mathcal{C}_{\text{bench}} = \left(\begin{bmatrix} +1 & +1 & +1 & -1 & +1 & +1 & -1 & +1 \\ +1 & -1 & +1 & +1 & +1 & -1 & -1 & -1 \end{bmatrix} \begin{bmatrix} +1 & +1 & +1 & -1 & -1 & -1 & +1 & -1 \\ +1 & -1 & +1 & +1 & -1 & +1 & +1 & +1 \end{bmatrix} \right). \quad (7)$$

As can be seen, there are $J = 2$ flocks of codes in $\mathcal{C}_{\text{bench}}$, each flock contains $M = 2$ codes, and the length of each code is $N = 8$. It can be verified that $\mathcal{M}(\mathcal{C}_{\text{bench}}) = 0$.

To rediscover the code set, there are 32 symbols to be filled in the game, and the number of all possible sequence-set patterns is $2^{32} \approx \mathcal{O}(10^9)$. Discovering the global optimum out of $\mathcal{O}(10^9)$ possible patterns is, in fact, not a difficult problem based on brute-force exhaustive search (even though it takes several days on our computer). The results of exhaustive search indicate that $\mathcal{C}_{\text{bench}}$ in (7) is not the only optimal pattern when $J = 2$, $M = 2$, and $N = 8$. There are in total 384 optimal patterns (that achieves $\mathcal{M}^* = 0$) that can be divided into 12 nonisomorphic types (i.e., each pattern has 31 other isomorphic patterns, see our technical report [21] for the specific shapes of isomorphic patterns).

4) Implementation: We implemented and ran AlphaSeq on a computer with a single CPU (Intel Core i7-6700) and a single GPU (NVIDIA GeForce GTX 1080 Ti).[5] The parameter settings are listed in Table I.

For the symbol filling game, we set $K = MJ = 4$, $N = 8$, and $\ell = 4$. In other words, in each time step, four symbols were placed in the 4×8 sequence set, and an episode ended after $\lceil NK/\ell \rceil = 8$ time steps when we obtained a complete sequence set. The metric function and reward function were then calculated following (5) and (6). An episode gave us eioght experiences.

For DNN-guided MCTS, at each state s_i, we first set s_i as the root node υ_0, and then ran $q = 400$ look-ahead simulations starting from υ_0. For each simulation, Dirichlet noise $\mathrm{Dir}([\alpha_0, \alpha_1, \ldots, \alpha_{2^\ell - 1}])$ was added to the prior probability of υ_0 to introduce exploration, where the parameters for Dirichlet distribution are set as $\alpha_0 = \alpha_1 = \cdots = \alpha_{2^\ell - 1} = \alpha = 0.05$. After 400 simulations, the probabilistic move-selection policy $\mathbf{\Pi}(s_i)$ was then calculated by (22), where we set $\tau = 1$ for the

[5]Given the listed computation resource, another experiment is presented in our technical report [21] to study the best found sequence versus time consumption in the RL process of AlphaSeq.

TABLE I
HYPERPARAMETERS OF ALPHASEQ FOR COMPLEMENTARY CODE DISCOVERY

Items	Parameters	Definitions
The Designed Game	$K = 4$	Number of sequences in the target set
	$N = 8$	Length of each sequence
	$\ell = 4$	Number of symbols filled in each time step
MCTS	$q = 400$	Number of simulations in one MCTS
	$\alpha = 0.05$	Dirichlet noise
	$\tau = 10^{-4}$ or 1	Determines the way we calculate the move selection policy based on their visiting counts
DNN	$G = 100$	Every G episodes, the DNN is updated using the experiences accumulated in the latest $z \times G$ episodes
	$z = 3$	
	$K' = 4$	Width of input image
	$N' = 8$	Length of input image
	batch = 64	Mini-batch size

first one third time steps (the probability of choosing a move is proportional to its visiting counts), and $\tau = 10^{-4}$ for the rest of the time steps (deterministically choose the move with the most visiting counts).

The DNN implemented in AlphaSeq is a deep convolutional network (ConvNets). This DNN consists of six convolutional layers together with batch normalization and rectifier nonlinearities (detailed architecture of this ConvNets can be found in Appendix A). The DNN update cycle $G = 100$ and $z = 3$, that is, every $G = 100$ episodes, we trained the ConvNets using the experiences accumulated in the latest $z \times G = 300$ episodes (i.e., 2400 experiences) by stochastic gradient descent. In particular, the minibatch size was set to 64, and we randomly sampled $\lceil 2400/64 \rceil$ minibatches without replacement from the 2400 experiences to train the ConvNets. For each minibatch, the loss function is defined by (23) in Appendix A.

Remark: In Table I, the width and length of the input image fed into DNN are chosen to match with N and K, i.e., $K' = K = 4$ and $N' = N = 8$. However, it should be emphasized that this is not an absolute necessity. In general, we find that setting the input of the DNN to be an $\ell \times \lceil NK/\ell \rceil$ image can speed up the learning process of DNN. For example, if we had set $\ell = 5$ instead of $\ell = 4$ in this experiment, then it would better to set $K' = 5$ and $N' = 7$ (i.e., DNN takes an 5×7 image as input, and in each time step, one row of the image is filled). Accordingly, any intermediate state (i.e., a partially filled 4×8 sequence set pattern) must first be transformed to a 5×7 image before it is fed into the ConvNets (the last three symbols in the 5×7 set will be padded with 0 because the original 4×8 set has three fewer symbols).

D. Performance Evaluation

Over the course of training, AlphaSeq ran 8×10^3 episodes, in which 6.4×10^4 experiences were generated. To monitor the evolution of AlphaSeq, every $G = 100$ episodes when the DNN was updated, we evaluated the searching capability of AlphaSeq by using it (with the updated DNN) to play 50 noiseless games. In general, the more evaluation games AlphaSeq plays, the better their mean performance captures the performance of current AlphaSeq. As a balance, we let AlphaSeq play 50 games so that a smoothed mean performance curve can be obtained without excessive evaluation time. The mean metric $E[\mathcal{M}]$ and the minimum metric $\min[\mathcal{M}]$ of the 50 found sequence sets were recorded and plotted in Fig. 5. Note that the plots of metric and reward functions differ only in scale. We plot the metric evolution in Fig. 5 as it is a direct quality measurement of a sequence set in this specific application.

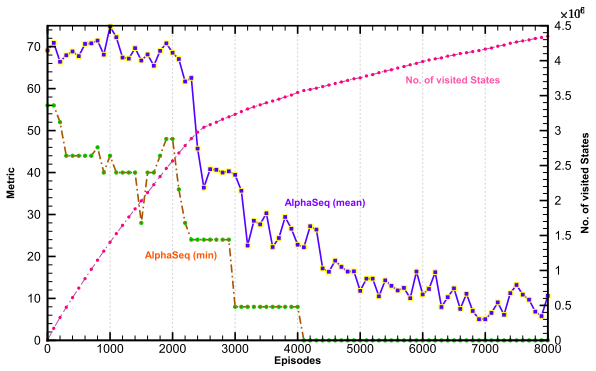


Fig. 5. RL process of AlphaSeq to rediscover a set of ideal complementary codes for MC-CDMA systems. Mean metric $E[\mathcal{M}]$, minimum metric $\min[\mathcal{M}]$, and the number of visited states versus episodes, where the DNN update cycle $G = 100$ and $z = 3$.

As can be seen from Fig. 5, with the continuous training of DNN, AlphaSeq gradually discovered sequence sets with smaller and smaller metric values. After 4100 episodes, AlphaSeq rediscovered an ideal complementary code set $\mathcal{C}_{\text{alpha}}$ given by

$$\mathcal{C}_{\text{alpha}} = \left(\begin{bmatrix} -1 & +1 & -1 & -1 & +1 & -1 & -1 & -1 \\ -1 & -1 & -1 & +1 & +1 & +1 & -1 & +1 \end{bmatrix} \right. \\ \left. \begin{bmatrix} +1 & -1 & +1 & +1 & +1 & -1 & -1 & -1 \\ +1 & +1 & +1 & -1 & +1 & +1 & -1 & +1 \end{bmatrix} \right). \tag{8}$$

It is straightforward to see that $\mathcal{C}_{\text{alpha}}$ is an isomorphic version to $\mathcal{C}_{\text{bench}}$: i.e., if we denote $\mathcal{C}_{\text{bench}}$ by $([\boldsymbol{a_1}, \boldsymbol{a_2}]^T, [\boldsymbol{b_1}, \boldsymbol{b_2}]^T)^T$, then $\mathcal{C}_{\text{alpha}} = ([-\boldsymbol{b_2}, -\boldsymbol{b_1}]^T, [\boldsymbol{a_2}, \boldsymbol{a_1}]^T)^T$. We specify that AlphaSeq could find different ideal sequence sets in different runs. For example, in another run, AlphaSeq eventually discovered a nonisomorphic ideal sequence set to $\mathcal{C}_{\text{bench}}$, giving

$$\mathcal{C}'_{\text{alpha}} = \left(\begin{bmatrix} +1 & -1 & -1 & -1 & -1 & -1 & +1 & -1 \\ -1 & +1 & +1 & +1 & -1 & -1 & +1 & -1 \end{bmatrix} \right. \\ \left. \begin{bmatrix} -1 & +1 & -1 & -1 & +1 & +1 & +1 & -1 \\ +1 & -1 & +1 & +1 & +1 & +1 & +1 & -1 \end{bmatrix} \right). \tag{9}$$

The complexity of AlphaSeq is measured by means of distinct states that have been visited. Specifically, we stored all the states (including intermediate states and terminal states) encountered over the course of training in a Hash table. Every G episodes, we recorded the length of the Hash table (i.e., the total number of visited states by then) and plotted them in Fig. 5 as the training goes on.

An interesting observation is that there is a turning point on the curve of the number of distinct visited states. The slope of this curve corresponds to the extent to which AlphaSeq is exploring new states in its choice of actions. Under the framework of AlphaSeq, there are two kinds of exploration as follows.

1) *Inherent Exploration:* This is introduced by the variance of the action-selection policy. That is, the more random the action-selection policy is, the more new states are likely to be explored by AlphaSeq.
2) *Artificial Exploration:* We deliberately add extra artificial randomness to AlphaSeq to let it explore more states.

For example, the Dirichlet noise added to the root vertex in DNN-guided MCTS, the temperature parameter τ that determines how to calculate the policy all add to the randomness. At the beginning of the game (i.e., episode 0), the policy of AlphaSeq is quite random inherently because the DNN is randomly initialized. Thus, both inherent exploration and artificial exploration contribute to the slope of this curve. At the end of the game (i.e., episode 8×10^3), the policy converges; hence, the inherent exploration drops off, and only artificial exploration remains.

This turning point was, in fact, observed in all simulations of AlphaSeq in various applications we tried (not just the application for rediscovering complementary code here; see Section IV on application of AlphaSeq to discover phase-coded sequences for pulse compression radar). In general, we can then divide the overall RL process of AlphaSeq into two phases based on this turning point. Phase I is an exploration-dominant phase (before the turning point), in which the behaviors of AlphaSeq are quite random. As a result, AlphaSeq actively explores increasingly more states per G episodes in the overall solution space. After gaining familiarity with the whole solution space, AlphaSeq enters an exploitation-dominant phase (after the turning point), in which instead of exploring for more states, AlphaSeq tends to focus more on exploitation.

Remark: The DNN update cycle G is important to guarantee that the algorithmic iteration proceeds in a direction of performance improvement. In AlphaSeq, given a DNN ψ_θ, the move-selection policy $\mathbf{\Pi}$ given by the ψ_θ-guided MCTS is usually much stronger than the raw policy output $\boldsymbol{P}$ of ψ_θ. Thus, we first run ψ_θ-guided MCTS to play G games and generate $\lceil NK/\ell \rceil \times G$ experiences. Then, we use these experiences to train a new DNN $\psi_{\theta'}$, so that $\psi_{\theta'}$ can learn the stronger move given by ψ_θ-guided MCTS.

In this context, the DNN update cycle G must be chosen so that the $\lceil NK/\ell \rceil \times G$ experiences are sufficient to capture the fine details of $\mathbf{\Pi}$ given by ψ_θ-guided MCTS. In particular, parameter G is closely related to ℓ: a larger ℓ means more elements in $\mathbf{\Pi}$ (i.e., $\mathbf{\Pi}$ must capture 2^ℓ possible moves in each step), and hence, a larger G is needed to guarantee that $\mathbf{\Pi}$ is well represented by the $\lceil NK/\ell \rceil \times G$ experiences.

As stated in Section II, the essence of AlphaSeq is a process of iterative "game-play with DNN-guided MCTS" and "DNN update": the improvement of DNN brings about improvement of the DNN-guided MCTS, and the experiences generated by the improved MCTS, in turn, bring about further improvement of the DNN through training. To verify this, each time when the DNN is updated, we assess the new DNN by using it (without MCTS, and no noise) to discover 50 sequences and record their mean metric $E[\mathcal{M}']$. Specifically, at each state s_i, the player directly adopts the raw policy output of the DNN, i.e., $\boldsymbol{P}(s_i)$, to sample the next move without relying on the MCTS outputs $\mathbf{\Pi}(s_i)$.

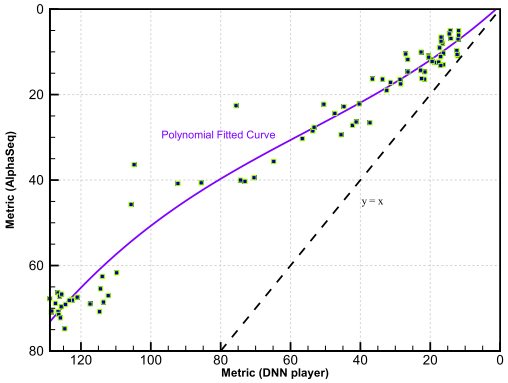


Fig. 6. Polynomial fit convergence curve for AlphaSeq and DNN player, where the DNN update cycle $G = 100$. The positive direction of the x-axis is a direction of the performance improvement for DNN, while the positive direction of the y-axis is a direction of the performance improvement for AlphaSeq.

Fig. 6 shows all the $(E[\mathcal{M}'], E[\mathcal{M}])$ pairs over the course of training and the corresponding polynomial fitted convergence curve. In particular, the positive direction of the x-axis in Fig. 6 is a direction of performance improvement for DNN, and the positive direction of y-axis is a direction of performance improvement for AlphaSeq. The convergence curve in Fig. 6 reflects how the two ingredients, "MCTS-guided game-play" and "DNN update," interplay and mutually improve in the RL process of AlphaSeq.

IV. AlphaSeq for Pulse Compression Radar

Radar radiates radio pulses for the detection and location of reflecting objects [4]. A classical dilemma in radar systems arises from the choice of pulse duration: given a constant power, longer pulses have higher energy, providing greater detection range; shorter pulses, on the other hand, have larger bandwidth, yielding a higher resolution. Thus, there is a tradeoff between distance and resolution. Pulse compression radar can enable high-resolution detection over a large distance [4], [22], [23]. The key is to use modulated pulses (e.g., phase-coded pulse) rather than conventional non-modulated pulses.

A. Pulse Compression Radar and Phase codes

The transmitter of a binary phased-coded pulse compression radar system transmits a pulse modulated by N rectangular subpulses. The subpulses are a binary phase code $\boldsymbol{s}$ of length N. Each entry of the code is $+1$ or -1, corresponding to phase 0 and π. Following the definition in [23] and [24], after subpulse-matched filtering (MF) and analog-to-digital conversion, the received sequence $\boldsymbol{y}$ is

$$\boldsymbol{y} = h_0 \boldsymbol{s} + \sum_{n=1-N, n\neq 0}^{N-1} h_n \boldsymbol{J_n} \boldsymbol{s} + \boldsymbol{w} \tag{10}$$

where: 1) $\{h_n : n = 1-N, 2-N, \ldots, N-2, N-1\}$ are coefficients proportional to the radar cross sections of different

range bins [23]. In particular, h_0 corresponds to the range bin of interest, and the radar's objective is to estimate h_0 given the received sequence $\boldsymbol{y}$; 2) $\boldsymbol{w}$ is the white Gaussian noise; and 3) matrix $\boldsymbol{J_n}$, as given in (11), is a shift matrix capturing the different propagation time needed for the clutter to return from different range bins [24].

$$J_i = \begin{array}{c} \text{Column: } \begin{array}{ccccccc} 0 & 1 & 2 & & i & & N-1 \end{array} \\ \begin{bmatrix} 0 & & \cdots & & 1 & \cdots & 0 \\ & 0 & & & & 1 & \\ & & 0 & & & & \vdots \\ & & & 0 & & & \\ \vdots & & & & \ddots & & 1 \\ & & & & & 0 & \vdots \\ & & & & & & 0 \\ 0 & & & \cdots & & & 0 \end{bmatrix} \end{array} \tag{11}$$

where $i = 1, 2, \ldots, N$–1 and $\boldsymbol{J}_{-i} = \boldsymbol{J}_i^T$. That is, in matrix $\boldsymbol{J_n}$, all entries except for that on the nth off-diagonal are 0. The effect of matrix $\boldsymbol{J_n}$ is to right-shift or left-shift the phase code $\boldsymbol{s}$ with zero padding: when $n < 0$, $\boldsymbol{J_n s}$ is a right-shifted version of $\boldsymbol{s}$; when $n > 0$, $\boldsymbol{J_n s}$ is a left-shifted version of $\boldsymbol{s}$.

To estimate the coefficient h_0, a widely studied estimator is the MF estimator [25]–[27]

$$\widehat{h}_0 = \frac{\boldsymbol{s}^T \boldsymbol{y}}{\boldsymbol{s}^T \boldsymbol{s}} = h_0 + \sum_{n=1-N, n\neq 0}^{N-1} h_n \frac{\boldsymbol{s}^T \boldsymbol{J_n s}}{\boldsymbol{s}^T \boldsymbol{s}} \tag{12}$$

where the additive white Gaussian noise (AWGN) is ignored since the received signal is interference-limited (i.e., the interference power dominates over the noise power). Given the fact that we have no information on $\{h_n : n \neq 0\}$, the problem is then to discover a phase code $\boldsymbol{s}$ that can maximize the SIR γ_{MF} (larger SIR yields better estimation performance)

$$\gamma_{\text{MF}} = \frac{(\boldsymbol{s}^T \boldsymbol{s})^2}{\sum_{n=1-N, n\neq 0}^{N-1} (\boldsymbol{s}^T \boldsymbol{J_n s})^2}. \tag{13}$$

In fact, this is the well-known "merit factor problem" occurring in various guises in many disciplines [26]. In the past few decades, a variety of phase codes have been devised to achieve large SIR (merit factor), e.g., the Rudin–Shapiro sequences (asymptotically, $\gamma_{\text{MF}} = 3$), m-sequences (asymptotically, $\gamma_{\text{MF}} = 3$), and Legendre sequences (asymptotically, $\gamma_{\text{MF}} = 6$) (see the excellent surveys [26], [27] and the references therein). Overall, the merit factor problem remains open. Experiment results show that γ_{MF} does not increase as the sequence length N increases. So far, the best-known merit factor of 14.08 is achieved by the Barker sequence of length 13.

The motivation of the MF estimator comes from the fact that MF provides the highest signal-to-noise ratio (SNR) in the presence of white Gaussian noise [28]. However, in the case of Radar, the received signal is interference-limited; hence, interference suppression is much more important. This motivates researchers to devise a MMF estimator [23], [24], [29].

Instead of using the transmitted phase-code $\boldsymbol{s}$, the MMF estimator uses a general real-valued code $\boldsymbol{x}$ to correlate the received sequence, giving

$$\widehat{h}_0 = \frac{\boldsymbol{x}^T \boldsymbol{y}}{\boldsymbol{x}^T \boldsymbol{s}} = h_0 + \sum_{n=1-N, n\neq 0}^{N-1} h_n \frac{\boldsymbol{x}^T \boldsymbol{J_n s}}{\boldsymbol{x}^T \boldsymbol{s}} \tag{14}$$

where the real-valued sequence $\boldsymbol{x}$ is to be optimized at the receiver. The problem is then to find a pair of sequences $(\boldsymbol{s}, \boldsymbol{x})$ so that the SIR γ_{MMF} in (15) can be maximized

$$\gamma_{\text{MMF}} = \frac{(\boldsymbol{x}^T \boldsymbol{s})^2}{\sum_{n=1-N, n\neq 0}^{N-1} (\boldsymbol{x}^T \boldsymbol{J_n s})^2}. \tag{15}$$

It had been shown in [24] that, given a phase code $\boldsymbol{s}$, the optimal sequence $\boldsymbol{x}$ that maximizes γ_{MMF} is $\boldsymbol{x}^* = \boldsymbol{R}^{-1}\boldsymbol{s}$, where matrix $\boldsymbol{R}$ is given by

$$\boldsymbol{R} = \sum_{n=1-N, n\neq 0}^{N-1} \boldsymbol{J_n s s}^T \boldsymbol{J_n}^T. \tag{16}$$

Substituting $\boldsymbol{x}^* = \boldsymbol{R}^{-1}\boldsymbol{s}$ in (15) gives

$$\gamma_{\text{MMF}} = \boldsymbol{s}^T \boldsymbol{R}^{-1} \boldsymbol{s}. \tag{17}$$

Note that γ_{MMF} only depends on the phase code $\boldsymbol{s}$; hence, the objective for the design of the MMF estimator is then to discover a phase-code $\boldsymbol{s}$ that can maximize γ_{MMF} in (17).

Remark: The MMF estimator is superior to the MF estimator since γ_{MMF} is not less than γ_{MF} given the same phase code $\boldsymbol{s}$. However, the problem of discovering a phase code $\boldsymbol{s}$ that maximizes (17) did not receive much attention from the research community compared with the merit factor problem [i.e., discovering a code $\boldsymbol{s}$ that maximizes (15)]. This is perhaps due to the more complex criterion and the lack of suitable mathematical tools [27].

In this section, we make use of AlphaSeq to discover phase codes for pulse compression radar with MMF estimator.

B. AlphaSeq for Pulse Compression Radar

We choose (17) as the metric function of AlphaSeq

$$\mathcal{M}(\boldsymbol{s}) = \boldsymbol{s}^T \boldsymbol{R}^{-1} \boldsymbol{s} \tag{18}$$

where matrix $\boldsymbol{R}$ is given in (16). The objective of AlphaSeq is then to discover the sequence that can maximize this metric function.

Given a phase code $\boldsymbol{s}$ with metric $\mathcal{M}(\boldsymbol{s})$, the linear reward function is defined as follows:

$$\mathcal{R}(\boldsymbol{s}) = \frac{2\mathcal{M}(\boldsymbol{s}) - \mathcal{M}_u - \mathcal{M}_l}{\mathcal{M}_u - \mathcal{M}_l} \tag{19}$$

where $\mathcal{M}_u$ and $\mathcal{M}_l$ are the upper and lower bounds for the search range of $\mathcal{M}(\boldsymbol{s})$. In general, we could set $\mathcal{M}_u = \max_{\boldsymbol{s}} \mathcal{M}(\boldsymbol{s})$ and $\mathcal{M}_l = \min_{\boldsymbol{s}} \mathcal{M}(\boldsymbol{s})$.

Remark: In Appendix C of our technical report [21], the value ranges of $\mathcal{M}(\boldsymbol{s})$ are derived as $\max_{\boldsymbol{s}} \mathcal{M}(\boldsymbol{s}) = 37$ and $\min_{\boldsymbol{s}} \mathcal{M}(\boldsymbol{s}) = (16/9N^3)$. However, we empirically find that, if we directly set the search range $\mathcal{M}_u = 37$ and $\mathcal{M}_l = (16/9N^3)$, AlphaSeq will be trapped in the exploration-dominant phase for a long time. This is because $\mathcal{M}_u - \mathcal{M}_l$ is too large. In other words, we are asking AlphaSeq to search over a large solution space for $\boldsymbol{s}$ all at once. We will

TABLE II
HYPERPARAMETERS OF ALPHASEQ FOR PHASE CODE DISCOVERY

Items	Parameters	Definitions
The Designed Game	$K = 1$	Number of sequences in the target set
	$N = 59$	Length of each sequence
	$\ell = 5$	Number of symbols filled in each time step
MCTS	$q = 900$	Number of simulations in one MCTS
	$\alpha = 0.1$	Dirichlet noise
	$\tau = 10^{-4}$ or 1	Determines the way we calculate the move selection policy based on their visiting counts
DNN	$G = 300$	Every G episodes, the DNN is updated using the experiences accumulated in the latest $z \times G$ episodes
	$z = 2$	
	$K' = 5$	Width of input image
	$N' = 12$	Length of input image
	batch = 64	Mini-batch size

later introduce a technique dubbed "segmented induction" to induce AlphaSeq to zoomed-in view to a good solution. In essence, segmented induction uses a smaller range of $[(16/9N^3), 37]$, but progressively changes $\mathcal{M}_u$ and $\mathcal{M}_l$ as better $\mathcal{M}(\boldsymbol{s})$ is obtained (i.e., focus our search within a subspace of $\boldsymbol{s}$ each time, but progressively changing the focus of the subspace within which we search).

Based on the metric function and reward function defined above, we implemented AlphaSeq and trained DNN to discover a phase code for the MMF estimator. A Legendre sequence [13] is chosen as the benchmark.

1) Benchmarks: We choose the Legendre sequence of length $N = 59$ as our benchmark

$$\boldsymbol{s}_{\mathrm{L}} = \begin{bmatrix} +1 & +1 & -1 & +1 & +1 & -1 & +1 & -1 & +1 & -1 & -1 & -1 \\ +1 & -1 & +1 & +1 & -1 & +1 & +1 & +1 & -1 & +1 & -1 & +1 \\ -1 & -1 & +1 & -1 & -1 & +1 & +1 & +1 & -1 & +1 & +1 & +1 \\ +1 & -1 & -1 & +1 & +1 & +1 & +1 & +1 & -1 & -1 & -1 & -1 \\ -1 & +1 & +1 & -1 & -1 & -1 & -1 & +1 & -1 & -1 & -1 & \end{bmatrix}.$$

For the MMF estimator, this Legendre sequence yields SIR $\gamma_{\mathrm{MMF}} \approx 10.98$. For reference, $\boldsymbol{s}_{\mathrm{L}}$ yields a merit factor of $\gamma_{\mathrm{MF}} \approx 6.19$ when the MF estimator is used.

For the corresponding AlphaSeq game, there are 59 symbols to fill. The number of all possible sequence-set patterns is 2^{59}. The complexity of the exhaustive search for the global optimum is $\mathcal{O}(10^{18})$, and it would take more than one million years for our computer to find the optimal solution. In other words, the optimal solution of $\boldsymbol{s}$ when $N = 59$ is unavailable. In this context, the second benchmark we choose is a random search. For a random search, we randomly create 59-symbol sequences and record the maximum SIR obtained given a fixed budget of random trials.

2) Implementation: In the AlphaSeq implementation, the parameter settings are listed in Table II. As seen in the table, we aim to discover one sequence of length 59 wherein $K = 1$ and $N = 59$ in the AlphaSeq game. The number of symbols filled in each time step is set to $\ell = 5$, and the ConvNets takes 5×12 images as input. To feed an intermediate state (i.e., a partially filled 1×59 pattern) into the ConvNets, we first transform it to a 5×12 image (the missing 1 symbol will be padded with 0). A complete sequence is obtained after $\lceil NK/\ell \rceil = 12$ time steps, where 60 symbols are obtained. Then, we ignore the last symbol and calculate the metric function and reward function following (18) and (19). The DNN update cycle G is set to 300 and $z = 2$. That is, every $G = 300$ episodes, DNN will be updated using the experiences accumulated in the latest 600 episodes.

Given the huge solution space, it is challenging for our computer to train AlphaSeq to find the optimal solution. For one thing, each episode in this problem consumes much more time than the complementary code rediscovery problem in Section III, because of the larger number of MCTSs run in each episode and the larger number of simulations run in each MCTS. For another, the large solution space in this problem requires a massive number of exploration-dominant episodes so that AlphaSeq can visit enough number of states to gain familiarity with the whole solution space. As a result, the exploration phase will last a long time before AlphaSeq enters the exploitation phase. To tackle the above challenges, we use the following two techniques to accelerate the training process.

1) Make more efficient use of experiences. Every G episodes, we trained the DNN using the experiences accumulated in the latest zG episodes ($zG\lceil NK/\ell \rceil$ experiences in total) by stochastic gradient descent. In Section III, the minibatches were randomly sampled without replacement. That gave us $\lceil zG\lceil NK/\ell \rceil/64 \rceil$ minibatches (64 was the minibatch size). Here, we want to make more efficient use of experiences. To this end, every G episodes, we randomly sample $\lceil zG\lceil NK/\ell \rceil/64 \rceil \times 6$ minibatches with replacement from the latest $zG\lceil NK/\ell \rceil$ experiences to train the ConvNets.
2) *Segmented Induction:* This technique is particularly useful when the upper and lower bounds of the metric function span a large range, or when there is no way to bound the metric function. The essence of segmented induction is to segment the large range of the metric function to several small ranges and define the linear reward in small ranges rather than in a single large range. To be more specific, assuming a metric function with values within the range $[0, D]$. Then, rather than initializing the search range $\mathcal{M}_l = 0$ and $\mathcal{M}_u = D$ in (19), we segment $[0, D]$ to three small overlapping ranges[6] $[0, D/2]$, $[D/3, 2D/3]$, and $[D/2, D]$ and define the linear reward in these small ranges: in episode 0, we define the reward function in the first small range and initialize $\mathcal{M}_l = 0$ and $\mathcal{M}_u = D/2$. With the training of DNN, AlphaSeq is able to discover better and better sequences in the range $[0, D/2]$. When AlphaSeq discovers sequences with reward approaching 1 (i.e., the mean metric function of the found sequences approaches $D/2$), we then redefine the reward with the second range $[D/3, 2D/3]$. That is, we set $\mathcal{M}_l = D/3$, $\mathcal{M}_u = 2D/3$, and let AlphaSeq discovers sequences in the second small range. When AlphaSeq is able to discover sequences with reward approaching 1 again, we redefine the reward

[6] a) Nonoverlapping intervals are inadvisable. Experimental results show that AlphaSeq cannot learn well when using nonoverlapping intervals. b) The small ranges segmented here are for illustration purpose only. In general, we need to design the ranges according to the specifics in different problems.

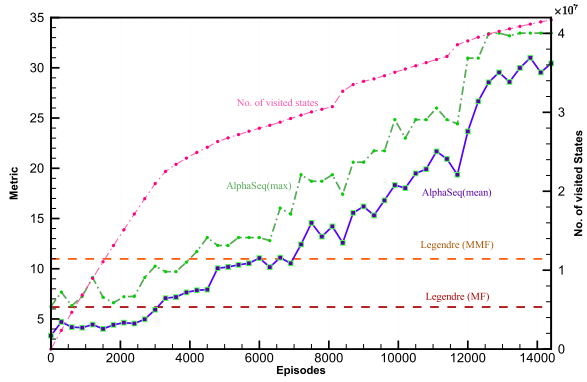


Fig. 7. RL process of AlphaSeq to discover a phase-coded sequence for pulse compression radar. Mean metric $E[\mathcal{M}]$, maximum metric $\max[\mathcal{M}]$, and a total number of visited states versus episodes, where the DNN update cycle $G = 300$ and $z = 2$.

in the third small range, and so on and so forth. Overall, with a smaller range at a given time, the slope of the reward function in (19) increases, allowing AlphaSeq to distinguish the relative quality of different sequences with higher contrast.

C. Performance Evaluation

For training, we ran AlphaSeq over 1.44×10^4 episodes, generating 1.73×10^5 experiences in total. As in Section III, to monitor the evolution of AlphaSeq, every $G = 300$ episodes when the DNN was updated, we evaluated the searching capability of AlphaSeq by using AlphaSeq (with the updated DNN) to play 50 noiseless games and recorded their mean metric $E[\mathcal{M}]$ and maximum metric $\max[\mathcal{M}]$. Fig. 7 shows the $E[\mathcal{M}]$ and $\max[\mathcal{M}]$ versus episodes during the process of RL.

As can be seen, the first 3300 episodes are the exploration-dominant phase and the episodes after that are the exploitation-dominant phase. After 1.26×10^4 episodes, AlphaSeq discovers a sequence with metric $\mathcal{M}(\boldsymbol{s}_{\text{alpha}}) \approx 33.45$

$$\boldsymbol{s}_{\text{alpha}} = \begin{bmatrix} +1 & +1 & +1 & +1 & +1 & +1 & +1 & +1 & +1 & +1 & +1 & +1 \\ +1 & +1 & +1 & +1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ +1 & +1 & +1 & -1 & -1 & +1 & +1 & -1 & +1 & +1 & -1 & +1 \\ -1 & -1 & +1 & -1 & +1 & -1 & +1 & -1 & -1 & +1 & -1 & +1 \\ -1 & +1 & -1 & +1 & -1 & +1 & -1 & +1 & -1 & +1 & -1 & \end{bmatrix}.$$

Compared with the Legendre sequence, $\boldsymbol{s}_{\text{alpha}}$ triples the SIR at the output of an MMF estimator.

Remark: In this implementation, the value range of $\mathcal{M}(\boldsymbol{s})$, i.e., $[16/9N^3, 37] \approx [0, 37]$, is segmented to three small ranges $[0, 15]$, $[5, 25]$, and $[10, 37]$. In the first 8100 episodes, the linear reward is defined in the first small range $[0, 15]$: metric 0 corresponds to reward -1, and 15 corresponds to reward 1; from episode 8101 to 11400, the linear reward is defined in the second small range $[5, 25]$; after episode 11401, the linear reward is defined in the last small range $[10, 37]$.

We next compare the searching capability of AlphaSeq with random search given the same complexity budget, where complexity is measured by the number of distinct visited states. For AlphaSeq, the visited states include both intermediate states and terminal states, while for a random search, only terminal states (i.e., completely filled sequences) will be searched.

In Fig. 8, the AlphaSeq curve is the maximal metric $\max[\mathcal{M}]$ versus the number of visited states. This curve is a transcription of two curves in Fig. 7: we combine the two curves, $\max[\mathcal{M}]$ versus episodes and number of visited states versus episodes, into one curve here. Fig. 8 also shows the maximal metric versus the number of visited states for random search.[7] To get this curve, given a state-visit budget, we performed 20 runs of the experiments. For each run i, we traced the maximum metric value obtained after a given number of random trials, denoted by $\mathcal{M}^i_{\max}(n_v)$, where n_v is the number of trials, which correspond to the number of visited (terminal) states. The black curve in Fig. 8 is $(1/20)\sum_i \mathcal{M}^i_{\max}(n_v)$ (i.e., a mean-max curve).

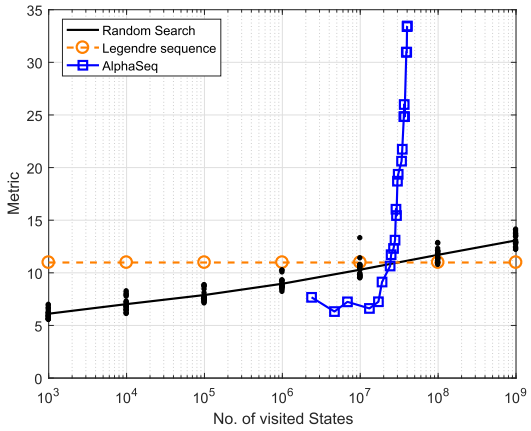


Fig. 8. Searching capability comparison of AlphaSeq and random search. The AlphaSeq curve is the maximal metric $\max[\mathcal{M}]$ versus the number of visited states.

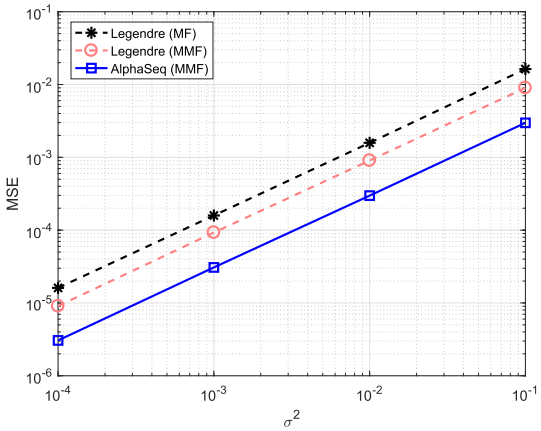


Fig. 9. MSE of $\boldsymbol{s}_{\text{alpha}}$ and $\boldsymbol{s}_{\text{L}}$ for h_0 estimation in pulse compression radar systems.

As can be seen from Fig. 8, the largest metric that random search can find is on average a log-linear function of the number visited states. After randomly visiting 10^8 states, the best sequence random search can find is on average with metric 11.71. On the other hand, AlphaSeq discovers sequences with $\max[\mathcal{M}] = 33.45$ after visiting only 4×10^7 states.

Finally, we assess the estimation performance of $\boldsymbol{s}_{\text{alpha}}$ benchmarked against the Legendre sequence $\boldsymbol{s}_{\text{L}}$ when used in a pulse compression radar system. In the simulation, we assume

[7] In our technique report [21], a deep Q-learning (DQL)-based approach, named DQLSeq, is developed to solve the MDP associated with the sequence discovery problem. The performance comparison between AlphaSeq and DQLSeq can be found in Appendix E of [21]. Overall, DQLSeq converges faster than AlphaSeq but is prone to getting stuck in local optimum.

that the radar radiates pulse internally modulated by $\boldsymbol{s}_{\text{alpha}}$ or $\boldsymbol{s}_{\text{L}}$. The received signal is given in (10), where $\{h_n\}$ are Gaussian random variables with zero mean and same variance σ^2, and AWGN noise is ignored. The receiver estimates h_0 using an MMF estimator, and we measure the estimation performance by MSE $\epsilon = (h_0 - \widehat{h_0})^2$. Fig. 9 shows MSE versus σ^2 for $\boldsymbol{s}_{\text{alpha}}$ and $\boldsymbol{s}_{\text{L}}$. As can be seen, $\boldsymbol{s}_{\text{alpha}}$ outperforms $\boldsymbol{s}_{\text{L}}$, and the MSE gains are up to about 5.23 dB.

V. Conclusion

This article has demonstrated the power of DRL for sequence discovery. We believe that sequence discovery by DRL is a good supplement to sequence construction by mathematical tools, especially for problems with complex objectives intractable to mathematical analysis.

Our specific contributions and results are as follows.

1) We proposed a new DRL-based paradigm, AlphaSeq, to algorithmically discover a set of sequences with the desired property. AlphaSeq leverages the DRL framework of AlphaGo to solve an MDP associated with the sequence discovery problem. The MDP is a symbol-filling game, where a player follows a policy to consecutively fill symbols in the vacant positions of a sequence set. In particular, AlphaSeq treats the intermediate states in the MDP as images and makes use of DNN to recognize them.
2) We introduced two new techniques in AlphaSeq to accelerate the training process. The first technique is to allow AlphaSeq to make ℓ moves at a time (i.e., filling ℓ sequence positions at a time). The choice of ℓ is a complexity tradeoff between the MCTS and the DNN. The second technique, dubbed segmented induction, is to change the reward function progressively to guide AlphaSeq to good sequences in its learning process.
3) We demonstrated the searching capabilities of AlphaSeq in two applications: 1) in MC CDMA systems, we used AlphaSeq to rediscover a set of ideal complementary codes that can zero-force all potential interferences and 2) in pulse compression radar systems, we used AlphaSeq to discover a new phase-coded sequence that triples the SIR at the output of a MMF estimator, benchmarked against the well-known Legendre sequence. The MSE gains are up to 5.23 dB for the estimation of radar cross sections.

Appendix A

This appendix describes the implementation details of AlphaSeq. Other than some custom features for our purpose, the general implementation follows AlphaGo Zero [12] and AlphaZero [16]. The source code can be found at GitHub [30].

A. MCTS

MCTS is performed at each intermediate state s_i to determine policy $\boldsymbol{\Pi}(s_i)$, and this is achieved by multiple look-ahead simulations along the tree. In the simulations, more promising vertices are visited frequently, while less promising vertices are visited less frequently. The problem is how to determine which vertices are more promising and which are less promising in the simulations, i.e., how to evaluate a vertex in MCTS. In standard MCTS algorithms, this vertex-evaluation is achieved by means of random rollouts. For example, for a new vertex encountered in each simulation, we run random rollout from this vertex to a leaf vertex such that a reward can be obtained (see [14] for more details). The randomly sampled rewards overall simulations are then used to evaluate a vertex.

In AlphaGo/AlphaSeq, instead of random rollouts, DNN is introduced to evaluate a vertex. The only two ingredients needed for MCTS are a root vertex v_0 and a DNN ψ_θ. First, given the root vertex v_0, a search tree can be constructed where each vertex contains 2^ℓ edges (since there are 2^ℓ possible moves for each state). Each edge, denoted by (v_i, a_j), $i = 0, 1, 2, \ldots,\ j = 0, 1, 2, \ldots, 2^\ell - 1$, stores three statistics: a visit count $N(v_i, a_j)$, a mean reward $Q(v_i, a_j)$, and an edge-selection prior probability $P(v_i, a_j)$. Second, MCTS uses DNN ψ_θ to evaluate each vertex (state). The input of ψ_θ is v_i and the output is $(\boldsymbol{P}, \mathcal{R}') = \psi_\theta(v_i)$. Specifically, each time we feed a vertex v_i into the DNN, it outputs a policy estimation $\boldsymbol{P}$ and a reward estimation $\mathcal{R}'$. Each entry in distribution $\boldsymbol{P}$ is exactly the prior probability $P(v_i, a_j)$ for each edge of vertex v_i, and $\mathcal{R}'$ will be used for updating the mean reward $Q(v_i, a_j)$, given by (21) later.

MCTS is operated by means of look-ahead simulations. Specifically, at a root vertex v_0, MCTS first initializes a "visited tree" (this visited tree is used to record all the vertices visited in the MCTS. It is initialized to have only one root vertex) and runs q simulations on the visited tree. Each simulation proceeds as follows [12]:

1) Select: All the simulations start from the root vertex v_0 and finish when a vertex that has not been seen is encountered for the first time. During a simulation, we always choose the edge that yields a maximum upper confidence bound. Specifically, at each vertex v_i, the simulation selects edge j^* to visit, and

$$j^* = \arg\max_j \left\{ Q(v_i, a_j) + c_p P(v_i, a_j) \frac{\sqrt{\sum_j N(v_i, a_j)}}{1 + N(v_i, a_j)} \right\}$$

where c_p is a constant controls the tradeoff between exploration and exploitation.

2) Expand and Evaluate: When encountering a previously unseen vertex v_L (for the first simulation, this v_L is, in fact, v_0), the simulation evaluates it using DNN, giving, $(\boldsymbol{P}_L, \mathcal{R}'_L) = \psi_\theta(v_L)$, where the policy distribution $\boldsymbol{P}_L = \{P_L(j) : j = 0, 1, 2, \ldots, 2^\ell - 1\}$. Then, we add this new vertex v_L to the visited tree, and the statistics of v_L's edges are initialized by $N(v_L, a_j) = 0$, $Q(v_L, a_j) = 0$, and $P(v_L, a_j) = P_L(j)$ for $j = 0, 1, 2, \ldots, 2^\ell - 1$.

3) Backup: After adding vertex v_L to the visited tree, the simulation updates all the vertices along the trajectory of encountering v_L. Specifically, for each edge (v_i, a_j) on the trajectory (including v_L), we update

$$N(v_i, a_j) = N(v_i, a_j) + 1 \tag{20}$$

$$Q(v_i, a_j) = Q(v_i, a_j) - \frac{Q(v_i, a_j) - \mathcal{R}'_L}{N(v_i, a_j)}. \tag{21}$$

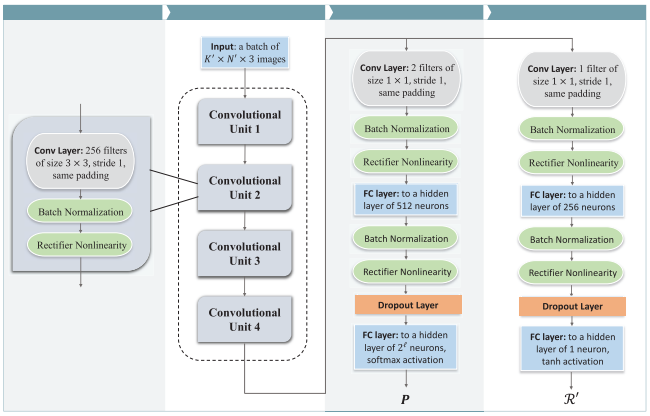


Fig. 10. Deep ConvNets implemented in AlphaSeq. This ConvNets consists of six convolutional layers together with batch normalization and rectifier nonlinearities.

After q simulations, MCTS then outputs a move selection probability for root vertex v_0 by

$$\mathbf{\Pi}(v_0) = \text{softmax}\left\{\frac{1}{\tau}\log N(v_0, a_j)\right\}. \tag{22}$$

For example, the move selection probability is determined by the visiting counts of the root vertex's edges. Parameter τ is a temperature parameter as in AlphaGo Zero [12]. In an episode, we set $\tau = 1$ (i.e., the move-selection probability is proportional to the visiting counts of each edge, yielding more exploration) for the first one third time steps and $\tau = 10^{-4}$ (deterministically choose the move that has the most visiting counts) for the rest of the time steps.

In the training iteration, when we play games to provide experiences for DNN, the Dirichlet noise, i.e., Dir($[\alpha_0, \alpha_1, \ldots, \alpha_{2^\ell-1}]$) with positive real parameters $\alpha_0, \alpha_1, \ldots, \alpha_{2^\ell-1}$, is added to the prior probability of root node v_0 to guarantee additional exploration. Thus, these games are called noisy games. Accordingly, there are noiseless games, in which the Dirichlet noise is removed. Generally, we play noiseless games to evaluate the performance of AlphaSeq with a trained DNN.

B. DNN

The DNN implemented in AlphaSeq is a deep convolutional network (ConvNets). This ConvNets consists of six convolutional layers together with batch normalization and rectifier nonlinearities, the details of which are shown in Fig. 10.

1) Input: The ConvNets takes $K' \times N' \times 3$ image stack as input. For a state s_i (i.e., an $K \times N$ partially filled sequence-set pattern), we first transform it to a $K' \times N' \times 1$ image (in general, we set $K' = \ell$ and $N' = \lceil NK/\ell \rceil$; zero-padding if $K'N' > KN$), and then perform feature extraction to transform it to a $K' \times N' \times 3$ image stack.

2) Feature Extraction: Feature extraction is a process to transform a $K' \times N' \times 1$ image to a $K' \times N' \times 3$ image stack comprising three binary feature planes. The three binary feature planes are constructed as follows. The first plane, X_1, indicates the presentence of "1" in the $K' \times N' \times 1$ image: $X_1(i, j) = 1$ if the intersection (i, j) has value "1" in the $K' \times N' \times 1$ image, and $X_1(i, j) = 0$ elsewhere. The second plane, X_2, indicates the presentence of "−1" in the $K' \times N' \times 1$ image: $X_2(i, j) = 1$ if the intersection (i, j) has value "−1" in the $K' \times N' \times 1$ image, and $X_2(i, j) = 0$ elsewhere. The third plane, X_3, indicates the presentence of "0" in the $K' \times N' \times 1$ image: $X_3(i, j) = 1$ if the intersection (i, j) has value "0" in the $K' \times N' \times 1$ image, and $X_3(i, j) = 0$ elsewhere.

3) Output: For each state s_i, DNN will output a policy estimation (i.e., a probability distribution) $\boldsymbol{P}(s_i) = (p_0, p_1, \ldots, p_{2^\ell-1})$ as the prior probability for the 2^ℓ edges of s_i, and a scalar estimation $\mathcal{R}' \in [-1, 1]$ on the expected reward of s_i.

4) Training: Every G games, we use the experiences accumulated in the most recent $z \times G$ games (i.e., $zG\lceil NK/\ell \rceil$ experiences) to update the DNN by stochastic gradient descent. The minibatch size is set to 64, and we randomly sample $\lceil zG\lceil NK/\ell \rceil/64 \rceil$ minibatches without replacement from the $zG\lceil NK/\ell \rceil$ experiences to train the ConvNets. For each minibatch, the loss function is defined to minimize the summation of mean-squared error and cross-entropy loss [12]

$$\mathcal{L} = (\mathcal{R} - \mathcal{R}')^2 - \mathbf{\Pi}^T \log \boldsymbol{P} + c\|\theta\|^2 \tag{23}$$

where the last term is L_2 regularization to prevent overfitting. Over the course of training, the learning rate is fixed to 10^{-4}.

APPENDIX B

This section derives the supremum of $\mathcal{M}(\mathcal{C})$ in (5) in Section III. Given the definitions of the correlation functions in (2)–(4), we first rewrite (5) as follows:

$$\mathcal{M}(\mathcal{C}) = \sum_{j_1=0}^{J-1}\sum_{j_2=j_1}^{J-1}\sum_{v=1}^{N-1}(|\text{CCF}_{j_1,j_2}[v]| + |\text{PCF}_{j_1,j_2}[v]|) + \sum_{j_1=0}^{J-1}\sum_{j_2=j_1+1}^{J-1}|\text{CCF}_{j_1,j_2}[0]|. \tag{24}$$

For the second term in (24), we have

$$\sum_{j_1=0}^{J-1}\sum_{j_2=j_1+1}^{J-1}|\text{CCF}_{j_1,j_2}[0]| \le \frac{J(J-1)}{2}NM. \tag{25}$$

Moreover, the first term in (24) can be simplified as follows:

$$\begin{aligned}&\sum_{j_1=0}^{J-1}\sum_{j_2=j_1}^{J-1}\sum_{v=1}^{N-1}(|\text{CCF}_{j_1,j_2}[v]| + |\text{PCF}_{j_1,j_2}[v]|)\\ &\quad= \sum_{j_1=0}^{J-1}\sum_{j_2=j_1}^{J-1}\sum_{v=1}^{N-1}(|\alpha[v] + \beta[v]| + |\alpha[v] - \beta[v]|)\\ &\quad= \sum_{j_1=0}^{J-1}\sum_{j_2=j_1}^{J-1}\sum_{v=1}^{N-1}2\max(|\alpha[v]|, |\beta[v]|)\end{aligned} \tag{26}$$

where $\alpha[v] = \sum_{m=0}^{M-1}\sum_{n=0}^{N-v-1} c_{j_1}^m[n]c_{j_2}^m[n+v]$, and $\beta[v] = \sum_{m=0}^{M-1}\sum_{n=N-v}^{N-1} c_{j_1}^m[n]c_{j_2}^m[n+v]$.

Note that set $\mathcal{C}$ is binary, hence, $\alpha[v]$ and $\beta[v]$ are summations of $(N-v)M$ and vM terms of 1 or −1, respectively. As a result, we have

$$\begin{aligned}\sum_{v=1}^{N-1}2\max(|\alpha[v]|, |\beta[v]|) &\le \sum_{v=1}^{N-1}2\max\{(N-v)M, vM\}\\ &= \begin{cases}\dfrac{3N^2-4N+1}{2}M, & \text{for } N \text{ odd}\\ \dfrac{3N^2-4N}{2}M, & \text{for } N \text{ even.}\end{cases}\end{aligned}$$

Finally, we can bound $\mathcal{M}(\mathcal{C})$ from (25) and (26) as follows:

$$\mathcal{M}(\mathcal{C}) \leq \begin{cases} \dfrac{(3N^2+1)(J^2+J)-2NJ(J+3)}{4}M, & \text{for } N \text{ odd} \\ \dfrac{3N^2(J^2+J)-2NJ(J+3)}{4}M, & \text{for } N \text{ even.} \end{cases}$$

The bounds can be achieved by an all -1 (or an all -1) sequence set, hence is tight.

REFERENCES

[1] OEIS. (1964). *The On-Line Encyclopedia of Integer Sequences*. [Online]. Available: https://oeis.org/A000040

[2] J. M. Heather and B. Chain, "The sequence of sequencers: The history of sequencing DNA," *Genomics*, vol. 107, no. 1, pp. 1–8, 2016.

[3] A. J. Viterbi, *CDMA: Principles of Spread Spectrum Communication*, vol. 122. Reading, MA, USA: Addison-Wesley, 1995.

[4] M. I. Skolnik, *Radar Handbook*. New York, NY, USA: McGraw-Hill, 2008.

[5] Y. Chen, Y.-H. Lo, K. W. Shum, W. S. Wong, and Y. Zhang, "CRT sequences with applications to collision channels allowing successive interference cancellation," *IEEE Trans. Inf. Theory*, vol. 64, no. 4, pp. 2910–2923, Apr. 2018.

[6] R. S. Sutton and A. Barto, *Reinforcement Learning: An Introduction*. Cambridge, MA, USA: MIT Press, 2018.

[7] M. L. Puterman, *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Hoboken, NJ, USA: Wiley, 2014.

[8] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.

[9] V. Mnih *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[10] D. Silver *et al.*, "Mastering the game of go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[11] Y. Li, "Deep reinforcement learning: An overview," 2017, *arXiv:1701.07274*. [Online]. Available: https://arxiv.org/abs/1701.07274

[12] D. Silver *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[13] M. Golay, "The merit factor of Legendre sequences (corresp.)," *IEEE Trans. Inf. Theory*, vol. IT-29, no. 6, pp. 934–936, Nov. 1983.

[14] C. B. Browne *et al.*, "A survey of Monte Carlo tree search methods," *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 1, pp. 1–43, Mar. 2012.

[15] L. Kocsis and C. Szepesvári, "Bandit based Monte-Carlo planning," in *Proc. Eur. Conf. Mach. Learn.* Berlin, Germany: Springer, 2006, pp. 282–293.

[16] D. Silver *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," 2017, *arXiv:1712.01815*. [Online]. Available: https://arxiv.org/abs/1712.01815

[17] H. H. Chen, *The Next Generation CDMA Technologies*. Hoboken, NJ, USA: Wiley, 2007.

[18] J. M. Velazquez-Gutierrez and C. Vargas-Rosales, "Sequence sets in wireless communication systems: A survey," *IEEE Commun. Surveys Tuts.*, vol. 19, no. 2, pp. 1225–1248, 2nd Quart., 2017.

[19] H.-H. Chen, J.-F. Yeh, and N. Suehiro, "A multicarrier CDMA architecture based on orthogonal complementary codes for new generations of wideband wireless communications," *IEEE Commun. Mag.*, vol. 39, no. 10, pp. 126–135, Oct. 2001.

[20] L. Welch, "Lower bounds on the maximum cross correlation of signals (Corresp.)," *IEEE Trans. Inf. Theory*, vol. IT-20, no. 3, pp. 397–399, May 1974.

[21] Y. Shao, S. C. Liew, and T. Wang, "AlphaSeq: Sequence discovery with deep reinforcement learning," 2018, [Online]. Available: https://arxiv.org/abs/1810.01218

[22] A. Boehmer, "Binary pulse compression codes," *IEEE Trans. Inf. Theory*, vol. IT-13, no. 2, pp. 156–167, Apr. 1967.

[23] R. M. Davis, R. L. Facnte, and R. P. Perry, "Phase-coded waveforms for radar," *IEEE Trans. Aerosp. Electron. Syst.*, vol. 43, no. 1, pp. 401–408, Jan. 2007.

[24] P. Stoica, J. Li, and M. Xue, "On sequences with good correlation properties: A new perspective," in *Proc. IEEE Inf. Theory Workshop (ITW)*, Jul. 2007, pp. 1–5.

[25] M. Golay, "The merit factor of long low autocorrelation binary sequences," *IEEE Trans. Inf. Theory*, vol. IT-28, no. 3, pp. 543–549, May 1982.

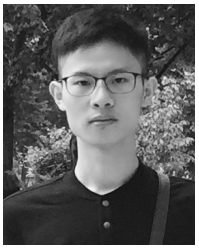
[26] T. Høholdt, "The merit factor problem for binary sequences," in *Proc. Int. Symp. Appl. Algebra, Algebr. Algorithms, Error-Correcting Codes (AAECC)*. Berlin, Germany: Springer, 2006, pp. 51–59.

[27] J. Jedwab, "What can be used instead of a Barker sequence?" *Contemp. Math.*, vol. 461, pp. 153–178, Feb. 2008.

[28] V.-P. Kaasila and A. Mammela, "Bit error probability of a matched filter in a Rayleigh fading multipath channel," *IEEE Trans. Commun.*, vol. 42, no. 234, pp. 826–828, Apr. 1994.

[29] M. H. Ackroyd and F. Ghani, "Optimum mismatched filters for sidelobe suppression," *IEEE Trans. Aerosp. Electron. Syst.*, vol. AES-9, no. 2, pp. 214–218, Mar. 1973.

[30] Y. Shao, S. C. Liew, and T. Wang. (2018). *AlphaSeq: Sequence Discovery with Deep Reinforcement Learning*. [Online]. Available: https://github.com/lintonshaw/AlphaSeq



Yulin Shao (S'17) received the B.E. and M.S. degrees from Xidian University (XDU), Xi'an, China, in 2013 and 2016, respectively. He is currently pursuing the Ph.D. degree with the Department of Information Engineering, The Chinese University of Hong Kong (CUHK), Hong Kong.

He was a Research Assistant with the Institute of Network Coding, CUHK, from March 2015 to August 2016. He was a Visiting Scholar with the Research Laboratory of Electronics, Massachusetts Institute of Technology, Cambridge, MA, USA, from September 2018 to March 2019. His current research interests include signal processing, fundamentals of wireless communications and networking, and machine learning (deep reinforcement learning, in particular).

Soung Chang Liew (S'84–M'87–SM'92–F'12) received the S.B., S.M., E.E., and Ph.D. degrees from the Massachusetts Institute of Technology (MIT), Cambridge, MA, USA.

From 1984 to 1988, he was with the MIT Laboratory for Information and Decision Systems, Cambridge, where he investigated fiber-optic communications networks. From March 1988 to July 1993, he was with Bellcore (now Telcordia), Piscataway, NJ, USA, where he engaged in broadband network research. Since 1993, he has been a Professor with the Department of Information Engineering, The Chinese University of Hong Kong (CUHK), Hong Kong. He is currently the Division Head of the Department of Information Engineering and a Co-Director of the Institute of Network Coding with CUHK. He is also serving as a Board Member for the Hong Kong Applied Science and Technology Institute (ASTRI), Hong Kong. He holds 15 U.S. patents. His current research interests include wireless networks, Internet of Things, intelligent transport systems, Internet protocols, multimedia communications, and packet switch design.

Dr. Liew is a fellow of IET and HKIE. He was a recipient of the first Vice-Chancellor Exemplary Teaching Award in 2000 and the Research Excellence Award in 2013 at the Chinese University of Hong Kong.

Taotao Wang (M'16) received the B.S. degree in electrical engineering from the University of Electronic Science and Technology of China, Chengdu, China, in 2008, the M.S. degree in information and signal processing from the Beijing University of Posts and Telecommunications, Beijing, China, in 2011, and the Ph.D. degree in information engineering from The Chinese University of Hong Kong (CUHK), Hong Kong, in 2015.

From 2015 to 2016, he was a Post-Doctoral Research Fellow with the Institute of Network Coding, CUHK. He joined the College of Information Engineering, Shenzhen University, Shenzhen, China, as an Assistant Professor. His current research interests include wireless communications and networking, statistical signal and data processing, and blockchain networks.

Dr. Wang was a recipient of the Hong Kong Ph.D. Fellowship.