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 paper 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, 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 CDMA systems. 2) AlphaSeq discovers new sequences that triple the signal-to-interference ratio – benchmarked against the well-known Legendre sequence – of a mismatched filter estimator in pulse compression radar systems.

Index Terms

Deep reinforcement learning, deep neural network, monte-carlo tree search, AlphaGo, multi-carrier CDMA, pulse compression radar.

Y. Shao, S. C. Liew and T. Wang are with the Department of Information Engineering, The Chinese University of Hong Kong, Shatin, New Territories, Hong Kong (e-mail: {sy016, soung, ttwang}@ie.cuhk.edu.hk).

I. INTRODUCTION

A sequence is a list of elements arranged in a certain order. Prime numbers arranged in ascending order, for example, is 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 filed 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 solution by the available mathematical tools. Faced with this problem, sequence designers may do 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) [6]; 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 [7] known for its ability to derive solutions for Markov Decision Processes (MDPs) [8] through a learning process. A salient feature of RL is "learning from interactions". Fig. 1 illustrates the framework
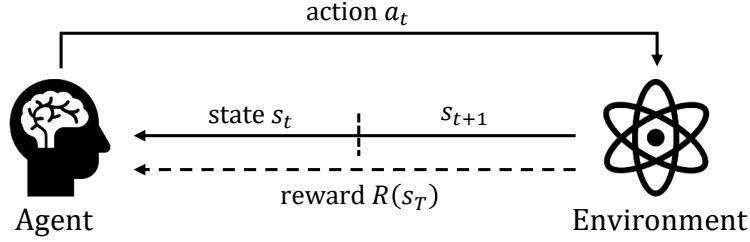
Fig. 1. In episodic reinforcement learning, the agent-environment interactions are broken into sessions called episodes. Each episode starts anew from an initial state s_0. The agent takes actions in successive discrete time steps $t = 0, 1, 2, ..., T - 1$, resulting in the state of the environment traversing through states $s_0, s_1, s_2, ...,$ until a terminal state s_T is reached, whereupon a reward $R(s_T)$ is given. The next episode begins independently of how the previous episode ended [7].

of episodic RL[1]. In the framework, an agent interacts with an environment in a sequence of discrete time steps $t = 0, 1, ..., T - 1$. 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 environment moving to state s_{t+1} in time step $t + 1$. The environment will feedback a reward $R(s_T)$ to the agent at the terminal state s_T, i.e., 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 generally 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 $P(a_t \mid s_t)$. The objective of the agent is to learn an expected-reward maximizing policy after going through multiple episodes. 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 [9] into the RL framework [10]–[12]. RL that makes use of deep neural networks (DNNs) to approximate the optimal policy function – directly or indirectly – is referred to as deep reinforcement learning (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,

---

[1]Instead of episodic RL, the agent-environment interaction in RL can also be non-episodic. In this case, the RL interactions go on indefinitely without an end state. A reward is given in each time step rather than at the end of an episode. We focus on episodic RL because it fits the problem of sequence discovery better.

and autonomous vehicle steering (see the excellent survey in [12]) have demonstrated its power in solving complex problems that thwart conventional approaches.

This paper 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:

- 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 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.

- AlphaSeq treats each intermediate state with some sequence positions filled and others vacant as an image. Each position is a pixel of the image. Given an input image (state), AlphaSeq makes use of a DNN to approximate the optimal policy that maximizes the reward. AlphaSeq uses a DRL framework similar to that of AlphaGo [13], in which DNN-guided MCTS (Monte-Carlo Tree Search [14]) 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.

- We introduce two techniques in AlphaSeq that are absent in AlphaGo. 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, learns to become an expert player.

We demonstrate the capability of AlphaSeq to discover two types of sequences:

1) We use AlphaSeq to rediscover a set of complementary codes for multi-carrier CDMA systems. In this application, AlphaSeq aims to discover a sequence set for which potential interferences in the multi-carrier CDMA system can be cancelled 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.

2) 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 output with high signal-to-interference ratio (SIR). The optimal sequences for MMF are not known and there is currently no known sequence that are provably optimal when the sequence is large. Benchmarked against the Legendre sequence [15], 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 paper is organized as follows. Section II formulates the sequence discovery problem and outlines the DRL framework of AlphaSeq. Section III and IV present the applications of AlphaSeq in multi-carrier CDMA systems and pulse compression radar systems, respectively. Section V concludes this paper. Throughout the paper, 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., $\{\boldsymbol{c}_k : k = 0, 1, ..., K-1\}$, where the k-th sequence is given by $\boldsymbol{c}_k = (c_k[0], c_k[1], ..., c_k[N-1])$. Each symbol of the sequences in $\mathcal{C}$ (i.e., $c_k[n]$) is drawn from a discrete alphabet $\mathcal{A}$. Without loss of generality, this paper focuses on binary sequences. That is, alphabet $\mathcal{A}$ 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 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 a 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 so forth. This rule will be used throughout the paper 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 $\mathcal{M}(\mathcal{C})$. It is the player's objective to learn a policy that makes sequential decisions to maximizes the reward, as more and more games are played.

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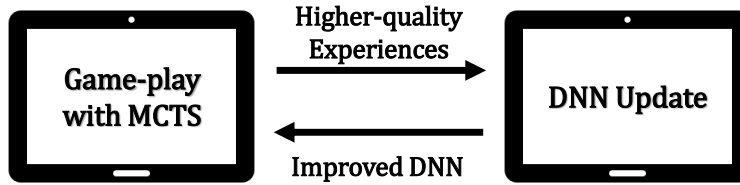
Fig. 2. The iterative algorithmic framework of AlphaGo/AlphaSeq. Improved DNN promotes the MCTS so that "game-play" generates experiences with higher quality; higher quality experiences can further enhance the DNN.

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). 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.

A successful combination of DNN and MCTS has been demonstrated in AlphaGo [11], [13], [16], where the authors use DNN to assess the vertices during the MCTS simulation, as opposed to using random rollouts in standard MCTS[2]. In this paper, 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, "game-play with MCTS" provides experiences to train the DNN so that the DNN can improves 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

[2]More details on the standard MCTS can be found in [14]. Throughout the paper, when we refer to MCTS, we mean the DNN-guided MCTS rather than the standard MCTS.

[3]AlphaGo itself is evolving, the DRL framework in this paper is based on AlphaGo Zero [13] and AlphaZero [16].
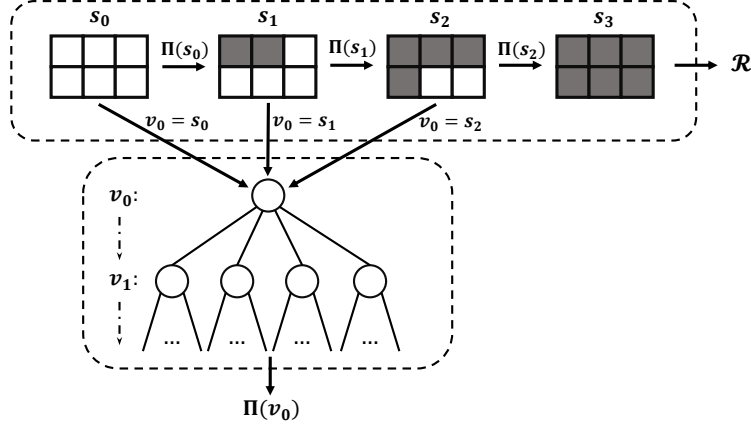
Fig. 3. An episode of game, where $K = 2$, $N = 3$, and $\ell = 2$. The NK positions are represented by the coloured squares: grey means that the positions are filled while white means that the positions are vacant. At each time step, following the output Π of MCTS, the player fills ℓ positions with value 1 or -1.

higher quality experiences to train the DNN. Through an iterative process, the MCTS and the DNN mutually enhances each other in a progressive manner over an underlying reinforcement learning 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.

Input and output of DNN -- The DNN is designed to estimate the value function and policy function of an intermediate state[4]. 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).

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: grey squares mean that the position has already been filled (with value 1 or

---

[4]The DNN will only evaluate intermediate states, but not the terminal states. For terminal states, the value function is known to the player (i.e., the reward function), and there is no policy.

-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 $\mathbf{\Pi}(s_i)$ (not the raw policy output $\boldsymbol{P}$ 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 policy $\mathbf{\Pi}(s_i)$ is a distribution over the 2^ℓ possible moves, and is given by MCTS.

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., equations (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., $\mathbf{\Pi}(s_i) = (\pi_0, \pi_1, ..., \pi_{2^\ell-1})$, is generated from the visiting counts of the root vertex's children in MCTS at states s_i (i.e., equation (22)).

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.

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; 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[5]. 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, Dirichlet noise is added to the prior probability of root node v_0 to induce exploration, as that in AlphaGo [13]. These games are also called noisy games. Instead of noisy games, we can also play noiseless games in which 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 (i) play G episodes of noisy games with ψ_θ-guided MCTS to generate experiences, where ψ_θ is the current DNN; (ii) use experiences gathered in the latest $z \times G$ episodes of games to train for a new DNN $\psi_{\theta'}$; (iii) 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 Table I.

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

- In AlphaGo, the player can choose any of the legal positions (the Go board is 19×19) to place its black or white piece, owing to the rule of game Go [11]. As a result, the overall legal state number is 3^{NK} (in Go, $N = 19$ and $K = 19$; each position can have three possible states: occupied by no stones, a white stone, or a black stone). On the other hand, in AlphaSeq, the positions to place symbols are predetermined in each time step, and the player only needs to determine the values of the symbols in the ℓ predetermined positions. This restriction brings about two benefits: a) for MCTS, the overall legal state number is

---

[5]The policy $\mathbf{\Pi}(s_i)$ generated by MCTS is more powerful than the raw output $\boldsymbol{P}(s_i)$ of DNN [13]. Thus, $\mathbf{\Pi}(s_i)$ can be used to improve $\boldsymbol{P}(s_i)$.

TABLE I

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

reduced 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 time step t has 2^{tl} possible values; b) the simpler rule reduces the amount of knowledge[6] the DNN needs to acquire, hence the DNN is easier to train compared with that in AlphaGo.

- 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

---

[6]The knowledge that the DNNs in AlphaGo and AlphaSeq are supposed to learn is also different. In AlphaGo, the DNN needs to decide which position is more promising to place its stone; while in AlphaSeq, the DNN is supposed to decide which ℓ symbols to place in the next ℓ predetermined positions.

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.

- 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 [13]. 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.

- Compared with AlphaGo, our computational power is rather limited. Thus, for large sequence set beyond our computational power, a new technique, dubbed segmented induction, is devised to progressively discover better sequence set. We exhibit in Section IV that segmented induction performs well when applied to AlphaSeq.

In the following sections, we will demonstrate the searching capabilities of AlphaSeq in two applications: in Section III, we use AlphaSeq to rediscover an ideal complementary code set for multi-carrier CDMA systems; 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

Code division multiple access (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 pre-assigned to them [17]. 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], [18]. That is, 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 [19], [20]. However, legacy CDMA systems are self-jamming systems since their code sets cannot guarantee user
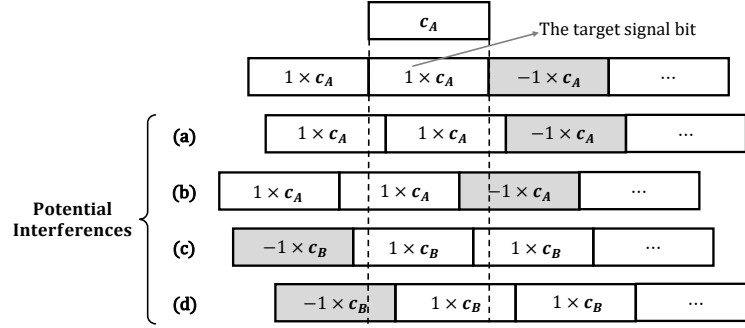
Fig. 4. The interferences caused by user asynchronies (misalignments of bit boundaries), multi-paths, and random signs of consecutive bits, in CDMA uplink. To decode user A's data, the receiver correlates the received signal with code c_A. Interferences are induced by (a) cyclic auto-correlation of c_A; (b) flipped auto-correlation of c_A; (c) cyclic cross-correlation between c_A and c_B; (d) flipped cross-correlation between c_A and c_B.

orthogonality under practical constraints and considerations, such as user asynchronies, multipath effects, and random signs of consecutive bits of user data streams [21][7].

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., (a) and (c) in Fig. 4); when the signs of the two interfering bits are different, the interferences are flipped correlation functions (i.e., (b) and (d) in Fig. 4). 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 simultane-

---

[7]In CDMA, "bit" refers to the baseband modulated information symbols (only BPSK/QPSK modulated symbols are considered in this paper, 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.

ously zero-forces the above correlation functions does not exist [22]. Code sets used in legacy CDMA systems trade-off among these correlation functions. For example, the m-sequence set has nearly ideal cyclic auto-correlation property (to be exact, the auto-correlation function of the m-sequence is -1 for any non-zero shift, hence is "nearly" optimal), while its cyclic cross-correlation and flipped correlation functions are unbounded. The Gold sequence set and the Kasami sequence set (candidate in W-CDMA) have better cyclic cross-correlation properties and acceptable cyclic auto-correlation properties, but their flipped correlations are unbounded (see the excellent survey [20] on the correlation functions of these sequences).

B. Multi-Carrier CDMA and Ideal Complementary Codes

The limitations of legacy CDMA systems motivate researchers to develop multi-carrier CDMA (MC-CDMA) systems where complementary codes can be used to simultaneously null all correlation functions among codes that may cause interferences [21], [23].

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, 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 de-spreads the received signal in each individual subcarrier (i.e., correlate the received signal in each sub-carrier with the corresponding element code), and sums up the de-spreading 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 a MC-CDMA system with J users, where a flock of M element codes of length N are assigned to each user. An ideal complementary code set $\mathcal{C} = \big\{ c_j^m[n] : j = 0, 1, ..., J-1; m = 0, 1, ..., M-1; n = 0, 1, ..., N-1 \big\}$ that can enable interference-free MC-CDMA systems is a code set that meets the following criteria simultaneously:

1) Ideal cyclic auto-correlation function (CAF): for the M element codes assigned to a user j, i.e., $\big\{ \boldsymbol{c}_j^m : m = 0, 1, ..., M-1 \big\}$, the sum of the cyclic auto-correlation function of each code is zero for any non-zero shift:

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

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

2) Ideal cyclic cross-correlation function (CCF): for two flocks of codes assigned to users j_1 and j_2, i.e., $\left\{ \boldsymbol{c}_{j_1}^m, \boldsymbol{c}_{j_2}^m : m = 0, 1, ..., M - 1 \right\}$, the sum of their cyclic cross-correlation functions is always zero irrespective of the relative shift:

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

where delay $v = 0, 1, 2, .., N - 1$ 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., $\left\{ \boldsymbol{c}_{j_1}^m, \boldsymbol{c}_{j_2}^m : m = 0, 1, ..., M - 1 \right\}$, the sum of their flipped correlation functions is always zero for any non-zero shift (flipped correlation is only defined for non-zero delay):

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

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

Some known mathematical constructions of ideal complementary codes are available in [18]. 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. Further, we would like to investigate the impact of the hyper parameters 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 subsection, 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), (3), and (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.

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

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

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 non-linear) 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 allow these problems to share the same underlying hyper parameters in DNN and MCTS of the AlphaSeq architecture). To rediscover the ideal complementary code, we define the reward function as follows:

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

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

where $\mathcal{M}_u$ is some sort of a worst-case $\mathcal{M}(\mathcal{C})$. That is, 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 $\mathcal{M}_u = \max_{\mathcal{C}} \mathcal{M}(\mathcal{C})$[8], and initialize the DNN to ψ_{θ_0} (i.e., the parameters in the DNN is 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 [18] is chosen as benchmark.

---

[8]See Appendix B for the derivation of $\max_{\mathcal{C}} \mathcal{M}(\mathcal{C})$.

TABLE II

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

Benchmark: When $J = 2$, $M = 2$, and $N = 8$, the ideal complementary code set exists. The mathematical constructions in [18] gives us

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

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 (that achieves $\mathcal{M}^* = 0$) when $J = 2$, $M = 2$, and $N = 8$. There are in fact 384 optimal patterns that can be divided into 12 non-isomorphic types (i.e., each pattern has 31 other isomorphic patterns, see Appendix B for the definition of isomorphic pattern).

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). The parameter settings are listed in Table. II.

For the symbol filling game, we set $K = MJ = 4$, $N = 8$, and $\ell = 4$. In other words, in each time step, 4 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 8 experiences.

For DNN-guided MCTS, at each state s_i, we first set s_i as the root node v_0, and then ran $q = 400$ look-ahead simulations starting from v_0. For each simulation, Dirichlet noise $\text{Dir}([\alpha_0, \alpha_1, ..., \alpha_{2^\ell - 1}])$ was added to the prior probability of v_0 to introduce exploration, where the parameters for Dirichlet distribution are set as $\alpha_0 = \alpha_1 = ... = \alpha_{2^\ell - 1} = \alpha = 0.05$. After 400 simulations, the probabilistic move-selection policy $\Pi(s_i)$ was then calculated by (22), where we set $\tau = 1$ for the 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 mini-batch size was set to 64, and we randomly sampled $\lceil 2400/64 \rceil$ mini-batches without replacement from the 2400 experiences to train the ConvNets. For each mini-batch, the loss function is defined by (23) in Appendix A.

Remark: In Table II, the width and length of the input image fed into DNN is 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 3 symbols in the 5×7 set will be padded with 0 because the original 4×8 set has 3 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
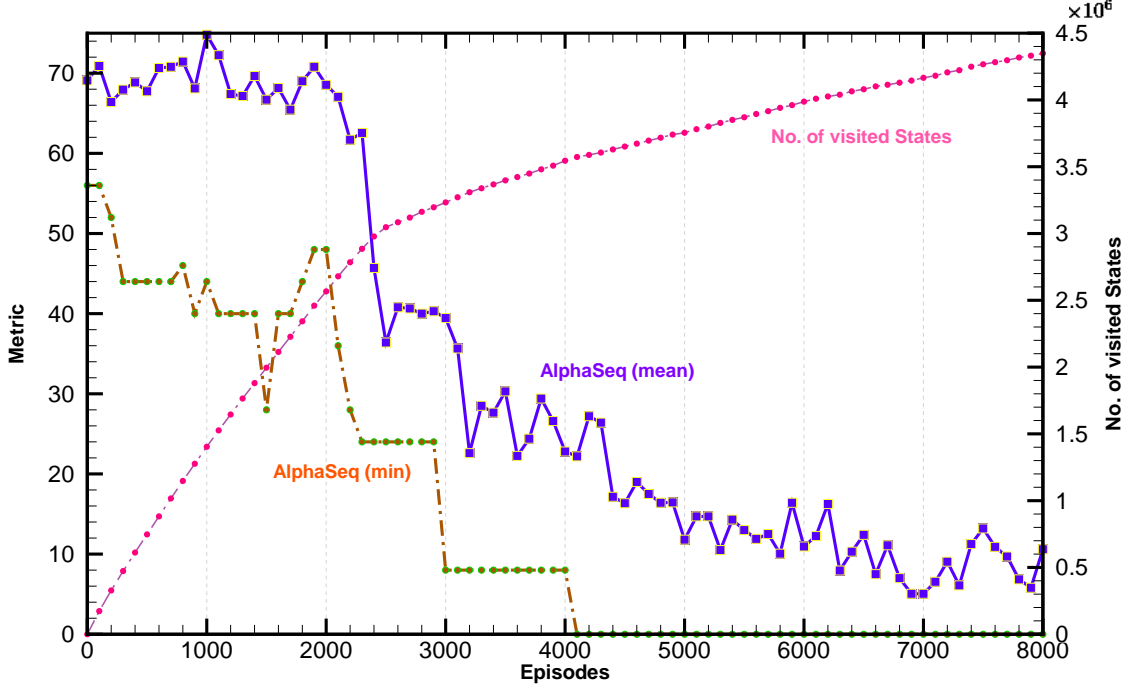
Fig. 5. The reinforcement learning process of AlphaSeq to rediscover a set of ideal complementary code for MC-CDMA systems. Mean metric $E[\mathcal{M}]$, minimum metric $\min[\mathcal{M}]$ and the number of visited states versus episodes, where the DNN update cycle $G = 100$ and $z = 3$.

DNN) to play 50 noiseless games (these 50 noiseless games are in addition to the $G = 100$ noisy games used to provide experiences to train the DNN). The mean metric $E[\mathcal{M}]$ and the minimum metric $\min[\mathcal{M}]$ of the found 50 sequence sets were recorded and plotted in Fig. 5.

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

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

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 found that AlphaSeq could find different ideal sequence set in different runs. For example, in another run, AlphaSeq

eventually discovered a non-isomorphic ideal sequence set to $\mathcal{C}_{\text{bench}}$, giving

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

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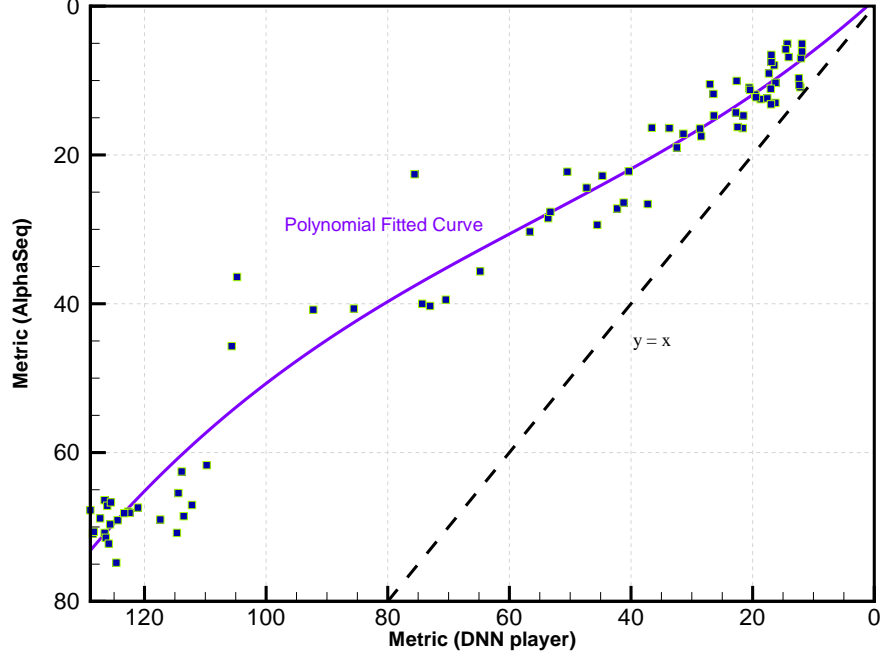
Fig. 6. The polynomial fitted convergence curve for AlphaSeq and DNN player, where the DNN update cycle $G = 100$. The positive direction of x-axis is a direction of performance improvement for DNN, while the positive direction of y-axis is a direction of performance improvement for AlphaSeq.

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 $\mathbf{P}$ 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 brings 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 $\boldsymbol{\Pi}(s_i)$.

Fig. 6 presents all the $(E[\mathcal{M}'], E[\mathcal{M}])$ pair in the exploitation phase, and the corresponding polynomial fitted convergence curve. In particular, the positive direction of 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 reinforcement learning 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 higher resolution. Thus, there is a trade-off between distance and resolution. Pulse compression radar can enable high-resolution detection over a large distance [4], [24], [25]. 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 [25] and [26], after subpulse-matched filtering 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) $\{h_n : n = 1-N, 2-N, ..., N-2, N-1\}$ are coefficients proportional to the radar cross sections of different range bins [25]. 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; 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 [26].

$$
\boldsymbol{J_i} = \begin{bmatrix} 0 & & \cdots & 1 & \cdots & 0 \\ & 0 & & & 1 & \\ & & 0 & & & \vdots \\ & & & 0 & & \ddots \\ \vdots & & & & \ddots & 1 \\ & & \ddots & & & 0 & \vdots \\ & & & & & & 0 \\ 0 & & \cdots & & & 0 & 0 \end{bmatrix} ,
$$

$$\text{Column:} \quad 0 \quad 1 \quad 2 \qquad i \qquad N-1$$

(11)

where $i = 1, 2, ..., N-1$ and $\boldsymbol{J_{-i}} = \boldsymbol{J_i^T}$. That is, in matrix $\boldsymbol{J_n}$, all entries except for that on the n-th 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 matched filtering (MF) estimator:

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

(12)

where the AWGN noise 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 signal-to-interference ratio (SIR) γ_{MF} (larger SIR yields better estimation performance):

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

(13)

In fact, this is the well-known "merit factor problem" occurring in various guises in many disciplines [27]–[29]. 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, $\gamma_{\mathrm{MF}} = 3$), m-sequences (asymptotically, $\gamma_{\mathrm{MF}} = 3$), and Legendre sequences (asymptotically, $\gamma_{\mathrm{MF}} = 6$) (see the excellent surveys [28], [29] 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 Baker sequence of length 13.

The motivation of the MF estimator comes from the fact that matched filtering provides the highest signal-to-noise ratio (SNR) in the presence of white Gaussian noise [30]. 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 mismatched filtering (MMF) estimator [25], [26], [31].

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

$$\widehat{h}_0 = \frac{x^T y}{x^T s} = h_0 + \sum_{n=1-N,n\neq0}^{N-1} h_n \frac{x^T J_n s}{x^T s}, \tag{14}$$

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

$$\gamma_{\text{MMF}} = \frac{(x^T s)^2}{\sum_{n=1-N,n\neq0}^{N-1}(x^T J_n s)^2}. \tag{15}$$

It had been shown in [26] that, given a phase code s, the optimal sequence x that maximizes γ_{MMF} is $x^* = R^{-1}s$, where matrix R is given by

$$R = \sum_{n=1-N,n\neq0}^{N-1} J_n s s^T J_n^T. \tag{16}$$

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

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

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

Remark: *The MMF estimator is superior to the MF estimator since γ_{MMF} is no less than γ_{MF} given the same phase code s. However, the problem of discovering a phase code s that maximizes* (17) *did not receive much attention from the research community compared with the merit factor problem (i.e., discovering a code s that maximizes* (15)*). This is perhaps due to the more complex criterion, and the lack of suitable mathematical tools [29].* 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:

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

where matrix $\boldsymbol{R}$ is given in (16). For AlphaSeq, the objective is to discover the sequence that can maximize this metric function. Additional analyses on the structure of this metric function can be found in Appendix C.

Given a phase code $\boldsymbol{s}$ with metric $\mathcal{M}(\boldsymbol{s})$, we define a linear reward function 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 an upper bound and a lower bound of $\mathcal{M}(\boldsymbol{s})$, giving, $\mathcal{M}_u = 37$ and $\mathcal{M}_l = \frac{16}{9N^3}$ (see the derivations and analyses in Appendix C).

Remark: Generally, if $\mathcal{M}_u - \mathcal{M}_l$ is large in a problem, that means we are asking AlphaSeq to search over a large solution space all at once. We found empirically that setting the $\mathcal{M}_l = \frac{16}{9N^3}$ and $\mathcal{M}_u = 37$ as above can result in AlphaSeq not being able to zoom in to a good solution within an acceptable time. We will later introduce a technique dubbed "segmented induction" to induce AlphaSeq to zoom in to a good solution. In essence, segmented induction uses a smaller range of $\mathcal{M}_u - \mathcal{M}_l$, but progressively changes $\mathcal{M}_u$ and $\mathcal{M}_l$ as better $\mathcal{M}(\boldsymbol{s})$ is obtained.

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 [15] is chosen as the benchmark.

Benchmark: 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 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 random search. For random search, we randomly create 59-symbol sequences and record the maximum SIR obtained given a fixed budget of random trials.

Implementation: In the AlphaSeq implementation, the parameter settings are listed in Table III. As seen in the table, we aim to discover one sequence of length 59 wherein $K = 1$ and $N = 59$

TABLE III

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

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 follows 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 mini-batches were randomly sampled without replacement. That gave us $\lceil zG\lceil NK/\ell \rceil/64 \rceil$ mini-batches (64 was the mini-batch size). Here, we want to make more efficient use of experiences. To this end, every G episodes, we randomly sample $\lceil zG\lceil NK/\ell \rceil/64 \rceil \times 6$ mini-batches 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 $\mathcal{M}_l = 0$ and $\mathcal{M}_u = D$ in (19), we segment $[0, D]$ to three small overlapping ranges[9] $[0, D/2]$, $[D/3, 2D/3]$, $[D/2, D]$, 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 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

[9]a) Non-overlapping intervals are inadvisable. Experimental results show that AlphaSeq cannot learn well when using non-overlapping 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.
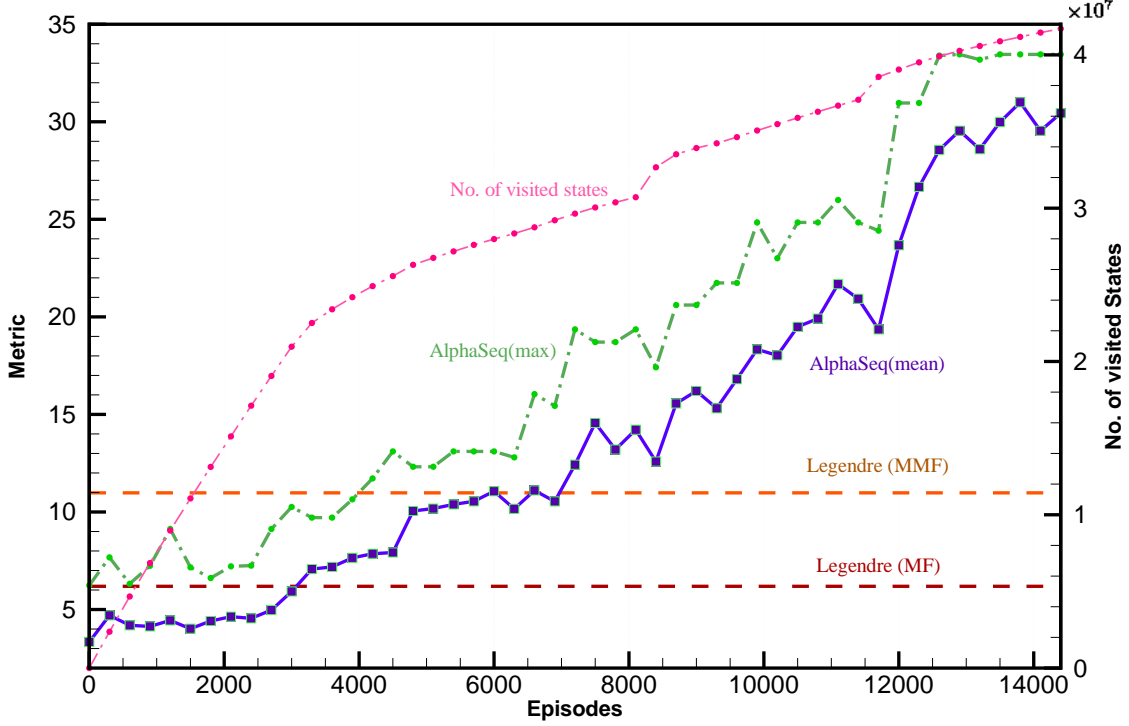
Fig. 7. The reinforcement learning process of AlphaSeq to discover a phase-coded sequence for pulse compression radar. Mean metric $E[\mathcal{M}]$, maximum metric $\max[\mathcal{M}]$ and total number of visited states versus episodes, where the DNN update cycle $G = 300$ and $z = 2$.

maximum metric $\max[\mathcal{M}]$. Fig. 7 presents the $E[\mathcal{M}]$ and $\max[\mathcal{M}]$ versus episodes during the process of reinforcement learning.

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 $\mathcal{M}(s_{\text{alpha}}) \approx 33.45$:

$$s_{\text{alpha}} = \begin{bmatrix} +1 & +1 & +1 & +1 & +1 & +1 & +1 & +1 & +1 & +1 & +1 & +1 \\ +1 & +1 & +1 & +1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ +1 & +1 & +1 & -1 & -1 & +1 & +1 & -1 & +1 & +1 & -1 & +1 \\ -1 & -1 & +1 & -1 & +1 & -1 & +1 & -1 & -1 & +1 & -1 & +1 \\ -1 & +1 & -1 & +1 & -1 & +1 & -1 & +1 & -1 & +1 & -1 & \end{bmatrix}.$$

Compared with the Legendre sequence, s_{alpha} triples the SIR at the output of a MMF estimator.

Remark: *In this implementation, the value range of $\mathcal{M}(s)$, 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*
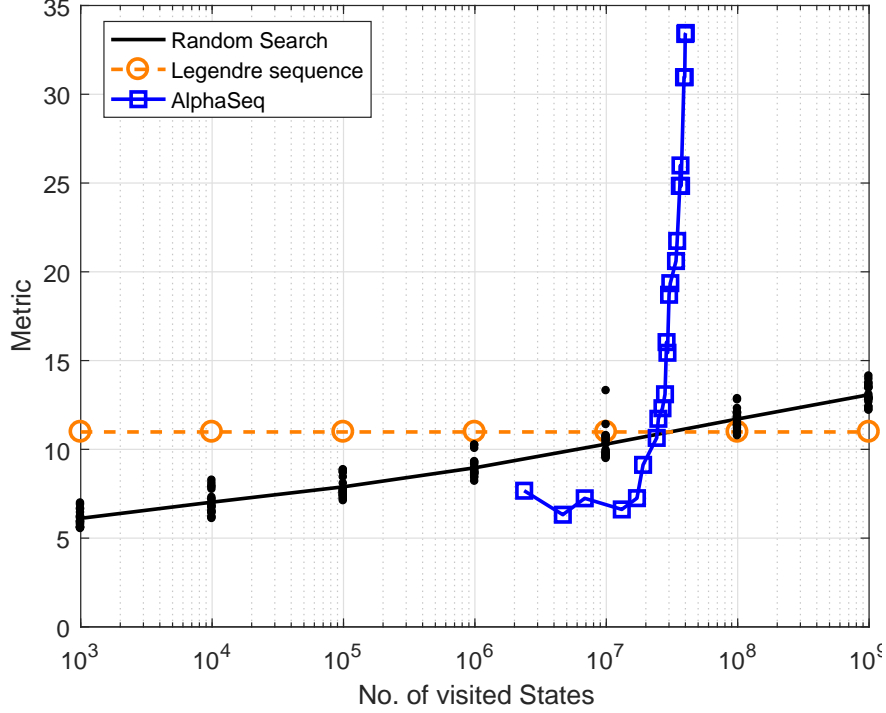
Fig. 8. The searching capability comparison of AlphaSeq and random search. The AlphaSeq curve is the maximal metric $\max[\mathcal{M}]$ versus number of visited states.

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[10]. For AlphaSeq, the visited states include both intermediate states and terminal states, while for 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 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 number of visited states for random search. 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 $\mathcal{M}^i_{\max}(n_v)$,

---

[10]Directly evaluating the complexity through the amounts of computation time consumed is not fair, because AlphaSeq uses both GPU and CPU in the implementation (and the CPU/GPU load varies over time) while random search uses only CPU (almost 100%).
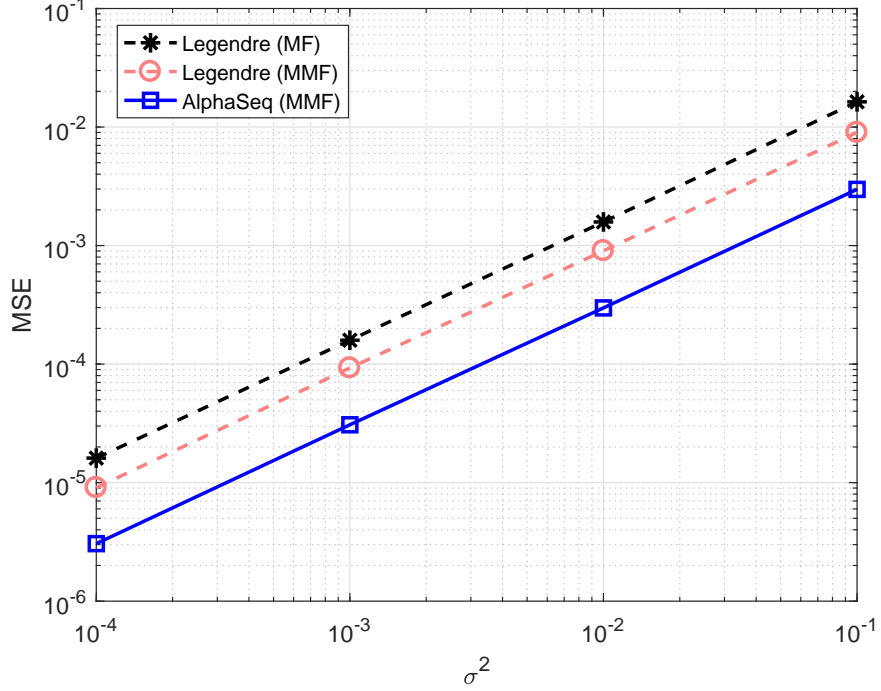
Fig. 9. The MSE of s_{alpha} and s_{L} for h_0 estimation in pulse compression radar systems.

where n_v is the number of trials, which correspond to the number of visited (terminal) states. The black curve in Fig. 8 is $\frac{1}{20} \sum_i \mathcal{M}^i_{\max}(n_v)$ (i.e., a mean-max curve).

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 s_{alpha} benchmarked against the Legendre sequence s_{L} when used in a pulse compression radar system. In the simulation, we assume the radar radiates pulse internally modulated by s_{alpha} or s_{L}. The received signal is given by equation (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 mean square error (MSE) $\epsilon = (h_0 - \widehat{h}_0)^2$. Fig. 9 presents MSE versus σ^2 for s_{alpha} and s_{L}. As can be seen, s_{alpha} outperforms s_{L}, and the MSE gains are up to about 5.23 dB.

V. Conclusion

This paper has demonstrated the power of deep reinforcement learning (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 desired property. AlphaSeq leverages the DRL framework of AlphaGo to solve a Markov Decision Process (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 deep neural network (DNN) to recognize them.

2) We introduced two new techniques absent in AlphaGo 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: a) We used AlphaSeq to redicover a set of ideal complementary codes that can zero-force all potentially interferences in multi-carrier CDMA systems. b) We used AlphaSeq to discover new sequences that triple the signal-to-interference ratio – benchmarked against the well-known Legendre sequence – of a mismatched filter estimator in pulse compression radar systems. The mean square error (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 [13] and AlphaZero [16]. The source code can be found at GitHub [32].

A. MCTS

MCTS is performed at each intermediate state s_i to determine policy $\mathbf{\Pi}(s_i)$, 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. That is, 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 over all 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), $i = 0, 1, 2, ...$, $j = 0$, $1, 2, ..., 2^\ell - 1$, 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 [13]:

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 $\boldsymbol{P}_L = \{P_L(j) : j = 0, 1, 2, ..., 2^\ell - 1\}$. 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 $j = 0, 1, 2, ..., 2^\ell - 1$.

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

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

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

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

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

That is, 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 [13]. 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, Dirichlet noise, i.e., $\mathrm{Dir}([\alpha_0, \alpha_1, ..., \alpha_{2^\ell - 1}])$ with positive real parameters $\alpha_0, \alpha_1, ..., \alpha_{2^\ell - 1}$, is added to the prior probability of root node v_0 to guarantee additional exploration. Thus, these games are called noisy games. Accordingly, there is noiseless games, in which Dirichlet noise is removed. Usually, 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 non-linearities, the details of which are shown in Fig. 10.

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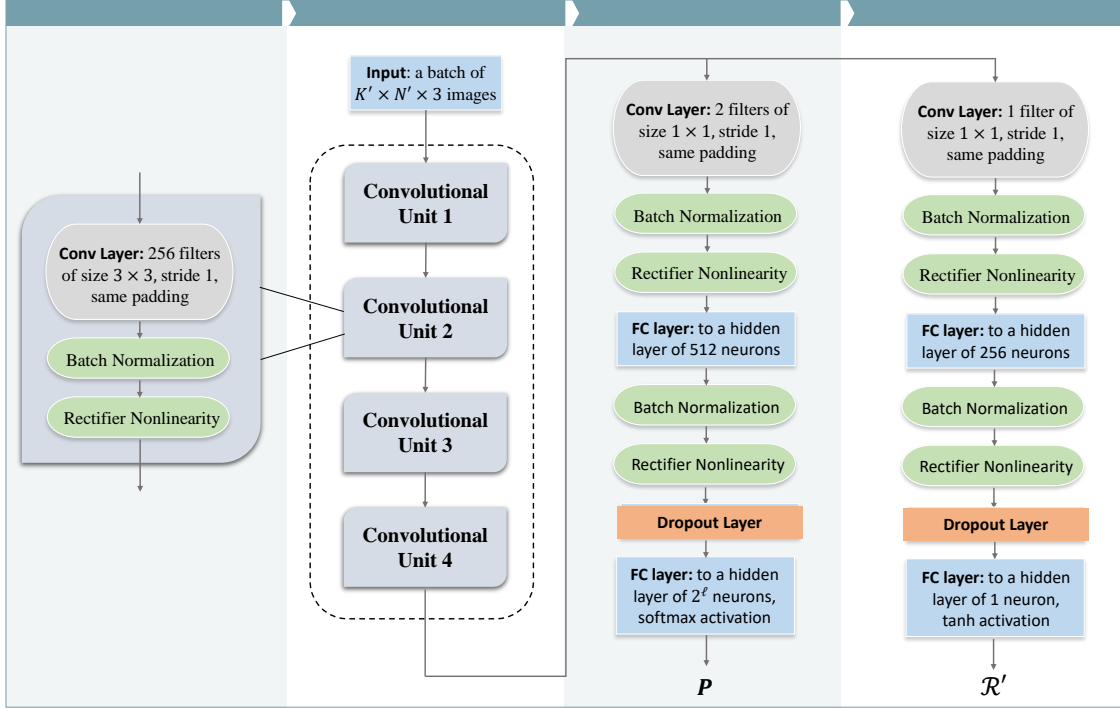
Fig. 10. The deep ConvNets implemented in AlphaSeq. This ConvNets consists of six convolutional layers together with batch normalization and rectifier nonlinearities.

(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.

- **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 3 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.

- **Output** -- For each state s_i, DNN will output a policy estimation (i.e., a probability distribution) $\boldsymbol{P}(s_i) = (p_0,\ p_1,\ ...,\ p_{2^\ell - 1})$ 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.

- **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 mini-batch size is set to 64, and we randomly sample $\lceil zG\lceil NK/\ell\rceil/64\rceil$ mini-batches without replacement from the $zG\lceil NK/\ell\rceil$ experiences to train the ConvNets. For each mini-batch, the loss function is defined to minimize the summation of mean-squared error and cross-entropy loss [13]

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

where the last term is L2 regularization to prevent overfitting. Over the course of training, the learning rate is fixed to 10^{-4}.

APPENDIX B

A. Supremum of the metric function

This subsection derives the supremum of $\mathcal{M}(\mathcal{C})$ in (5) in Section III. Given the definitions of the correlation functions in (2), (3), and (4), we first rewrite (5) as follows:

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

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

For the second term in (24), we have

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

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

$$\sum_{j_1=0}^{J-1} \sum_{j_2=j_1}^{J-1} \sum_{v=1}^{N-1} \left(\left|\mathrm{CCF}_{j_1,j_2}[v]\right| + \left|\mathrm{PCF}_{j_1,j_2}[v]\right|\right) = \sum_{j_1=0}^{J-1} \sum_{j_2=j_1}^{J-1} \sum_{v=1}^{N-1} \left(\left|\alpha[v] + \beta[v]\right| + \left|\alpha[v] - \beta[v]\right|\right)$$

$$= \sum_{j_1=0}^{J-1} \sum_{j_2=j_1}^{J-1} \sum_{v=1}^{N-1} 2 \max\left(\left|\alpha[v]\right|, \left|\beta[v]\right|\right), \tag{26}$$

where

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

Notice 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

$$\sum_{v=1}^{N-1} 2 \max\left(\left|\alpha[v]\right|, \left|\beta[v]\right|\right) \leq \sum_{v=1}^{N-1} 2 \max\left\{(N-v)M, vM\right\}$$

$$= \begin{cases} \frac{3N^2-4N+1}{2}M, & \text{for } N \text{ odd}, \\ \frac{3N^2-4N}{2}M, & \text{for } N \text{ even}. \end{cases} \qquad (27)$$

Finally, we can bound $\mathcal{M}(\mathcal{C})$ from (25), (26) and (27) as follows.

When N is odd, it follows directly that

$$\mathcal{M}(\mathcal{C}) \leq \frac{K(K+1)}{2} \times \frac{3N^2-4N+1}{2}M + \frac{K(K-1)}{2}NM = \frac{(3N^2+1)(K^2+K)-2NK(K+3)}{4}M.$$

When N is even,

$$\mathcal{M}(\mathcal{C}) \leq \frac{K(K+1)}{2} \times \frac{3N^2-4N}{2}M + \frac{K(K-1)}{2}NM = \frac{3N^2(K^2+K)-2NK(K+3)}{4}M.$$

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

In conclusion, we have

Proposition 1. *The supremum of* $\mathcal{M}(\mathcal{C})$ *is given by*

$$\sup \mathcal{M}(\mathcal{C}) = \begin{cases} \frac{(3N^2+1)(K^2+K)-2NK(K+3)}{4}M, & \text{for } N \text{ odd}, \\ \frac{3N^2(K^2+K)-2NK(K+3)}{4}M, & \text{for } N \text{ even}. \end{cases}$$

B. Isomorphic pattern

When $J = 2$, $M = 2$, and $N = 8$, each complementary code set has 31 other isomorphic patterns. For simplicity, let us denote a code set by $([\boldsymbol{a_1}, \boldsymbol{a_2}]^T, [\boldsymbol{b_1}, \boldsymbol{b_2}]^T)^T$, where $\boldsymbol{a_1}$ and $\boldsymbol{a_2}$ are the two element codes assigned to user A, while $\boldsymbol{b_1}$ and $\boldsymbol{b_2}$ are the two element codes assigned to user B. Then, the following code-set patterns are isomorphic.

1) Row-switching.

$$\left(\begin{bmatrix} \boldsymbol{a_1} \\ \boldsymbol{a_2} \end{bmatrix} \atop \begin{bmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \end{bmatrix} \right) ; \quad \left(\begin{bmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \end{bmatrix} \atop \begin{bmatrix} \boldsymbol{a_1} \\ \boldsymbol{a_2} \end{bmatrix} \right) ; \quad \left(\begin{bmatrix} \boldsymbol{a_2} \\ \boldsymbol{a_1} \end{bmatrix} \atop \begin{bmatrix} \boldsymbol{b_2} \\ \boldsymbol{b_1} \end{bmatrix} \right) ; \quad \left(\begin{bmatrix} \boldsymbol{b_2} \\ \boldsymbol{b_1} \end{bmatrix} \atop \begin{bmatrix} \boldsymbol{a_2} \\ \boldsymbol{a_1} \end{bmatrix} \right).$$

2) Sign-reversing.

$$\left(\begin{bmatrix} \boldsymbol{a_1} \\ \boldsymbol{a_2} \end{bmatrix} \atop \begin{bmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \end{bmatrix} \right) ; \quad \left(\begin{bmatrix} -\boldsymbol{a_1} \\ -\boldsymbol{a_2} \end{bmatrix} \atop \begin{bmatrix} \boldsymbol{b_1} \\ \boldsymbol{b_2} \end{bmatrix} \right) ; \quad \left(\begin{bmatrix} \boldsymbol{a_1} \\ \boldsymbol{a_2} \end{bmatrix} \atop \begin{bmatrix} -\boldsymbol{b_1} \\ -\boldsymbol{b_2} \end{bmatrix} \right) ; \quad \left(\begin{bmatrix} -\boldsymbol{a_1} \\ \boldsymbol{a_2} \end{bmatrix} \atop \begin{bmatrix} -\boldsymbol{b_1} \\ \boldsymbol{b_2} \end{bmatrix} \right) ;$$

$$\left(\begin{bmatrix} \boldsymbol{a_1} \\ -\boldsymbol{a_2} \end{bmatrix} \atop \begin{bmatrix} \boldsymbol{b_1} \\ -\boldsymbol{b_2} \end{bmatrix} \right) ; \quad \left(\begin{bmatrix} \boldsymbol{a_1} \\ -\boldsymbol{a_2} \end{bmatrix} \atop \begin{bmatrix} -\boldsymbol{b_1} \\ \boldsymbol{b_2} \end{bmatrix} \right) ; \quad \left(\begin{bmatrix} -\boldsymbol{a_1} \\ \boldsymbol{a_2} \end{bmatrix} \atop \begin{bmatrix} \boldsymbol{b_1} \\ -\boldsymbol{b_2} \end{bmatrix} \right) ; \quad \left(\begin{bmatrix} -\boldsymbol{a_1} \\ -\boldsymbol{a_2} \end{bmatrix} \atop \begin{bmatrix} -\boldsymbol{b_1} \\ -\boldsymbol{b_2} \end{bmatrix} \right).$$

Notice that $([\boldsymbol{a_1}, -\boldsymbol{a_2}]^T, [-\boldsymbol{b_1}, \boldsymbol{b_2}]^T)^T$ and $([-\boldsymbol{a_1}, \boldsymbol{a_2}]^T, [\boldsymbol{b_1}, -\boldsymbol{b_2}]^T)^T$ are isomorphic to $([\boldsymbol{a_1}, \boldsymbol{a_2}]^T, [\boldsymbol{b_1}, \boldsymbol{b_2}]^T)^T$. This is not straightforward at first sight. However, it can be seen

that $([\boldsymbol{a_1}, \boldsymbol{a_2}]^T, [\boldsymbol{b_1}, \boldsymbol{b_2}]^T)^T$ is isomorphic to $([-\boldsymbol{a_1}, \boldsymbol{a_2}]^T, [-\boldsymbol{b_1}, \boldsymbol{b_2}]^T)^T$, while $([-\boldsymbol{a_1}, \boldsymbol{a_2}]^T,$ $[-\boldsymbol{b_1}, \boldsymbol{b_2}]^T)^T$ is isomorphic to $([\boldsymbol{a_1}, -\boldsymbol{a_2}]^T, [-\boldsymbol{b_1}, \boldsymbol{b_2}]^T)^T$ and $([-\boldsymbol{a_1}, \boldsymbol{a_2}]^T, [\boldsymbol{b_1}, -\boldsymbol{b_2}]^T)^T$. Thus, $([\boldsymbol{a_1}, \boldsymbol{a_2}]^T, [\boldsymbol{b_1}, \boldsymbol{b_2}]^T)^T$ is also isomorphic to $([\boldsymbol{a_1}, -\boldsymbol{a_2}]^T, [-\boldsymbol{b_1}, \boldsymbol{b_2}]^T)^T$ and $([-\boldsymbol{a_1}, \boldsymbol{a_2}]^T, [\boldsymbol{b_1}, -\boldsymbol{b_2}]^T)^T$.

Overall, each sequence-set pattern has $4 \times 8 = 32$ isomorphic patterns.

APPENDIX C

This appendix analyzes the structure of the metric function $\mathcal{M}(\boldsymbol{s})$ in (18) in Section IV, and derives the upper and lower bounds for $\mathcal{M}(\boldsymbol{s})$. These two bounds are first derived mathematically. Then, we conjecture a tight upper bound for $\mathcal{M}(\boldsymbol{s})$. This conjectured bound can be achieved by a Barker sequence of length 13.

Lemma 2. *Matrix $\boldsymbol{R}$ and $\boldsymbol{R}^{-1}$ are positive definite matrices.*

Proof. By equation (16) in Section IV, $\boldsymbol{R}$ is the summation of $2N - 2$ symmetric matrices. Thus, $\boldsymbol{R}$ is symmetric, and hence $\boldsymbol{R}^{-1}$ is also symmetric ($\boldsymbol{R}$ is guaranteed to be non-singular [26]).

A symmetric matrix $\boldsymbol{A}$ is said to be positive definite (semi-definite) if 1) the scalar $\boldsymbol{x}^T \boldsymbol{A} \boldsymbol{x}$ is positive (nonnegative) for any non-zero column vector $\boldsymbol{x}$; or 2) all the eigenvalues of $\boldsymbol{A}$ are positive (nonnegative).

Given (16), we have

$$\boldsymbol{x}^T \boldsymbol{R} \boldsymbol{x} = \sum_{n=1-N, n \neq 0}^{N-1} \boldsymbol{x}^T \boldsymbol{J}_n \boldsymbol{s} \boldsymbol{s}^T \boldsymbol{J}_n^T \boldsymbol{x} \geq 0, \tag{28}$$

where the inequality follows since for each integer $n \in [1 - N, N - 1]$ and $n \neq 0$, $\boldsymbol{J}_n \boldsymbol{s} \boldsymbol{s}^T \boldsymbol{J}_n^T$ is positive semi-definite (In fact, for each individual n, $\boldsymbol{J}_n \boldsymbol{s} \boldsymbol{s}^T \boldsymbol{J}_n^T$ has $N - 1$ zero eigenvalues and one positive eigenvalue being $N - |n|$).

Given (28), $\boldsymbol{R}$ is positive semi-definite. Moreover, $\boldsymbol{R}$ has no zero eigenvalue since $\boldsymbol{R}$ is nonsingular. As a result, all the eigenvalues of $\boldsymbol{R}$ are positive, and hence $\boldsymbol{R}$ is guaranteed to be positive definite. Further, all the eigenvalues of $\boldsymbol{R}^{-1}$ are also positive, hence $\boldsymbol{R}^{-1}$ is also positive definite. $\square$

Lemma 3 (Weyl's inequalities [33]). *Given two $N \times N$ Hermitian matrices $\boldsymbol{A}$ and $\boldsymbol{B}$, and their eigenvalues $\lambda_1 \geq \lambda_2 \geq ... \geq \lambda_N$, $\mu_1 \geq \mu_2 \geq ... \geq \mu_N$, respectively. For another matrix $\boldsymbol{C} = \boldsymbol{A} + \boldsymbol{B}$, if we denote the eigenvalues of $\boldsymbol{C}$ by $v_1 \geq v_2 \geq ... \geq v_N$. Then,*

$$v_{i+j+1} \leq \lambda_{i+1} + \mu_{j+1},$$

$$v_{n-i-j} \geq \lambda_{n-i} + \mu_{n-j}.$$

Lemma 4 (Singular value bound [34]). *Let $\boldsymbol{A}$ be an $N \times M$ complex matrix, then its singular values are bounded by*

$$\sigma(\boldsymbol{A}) \leq \max_i \sum_j |a_{i,j}| \sum_k |a_{k,j}|. \tag{29}$$

where $i = 0, 1, ..., N-1$, $j = 0, 1, ..., M-1$, and $k = 0, 1, ..., N-1$.

Lemma 5 (Singular value bound [35]). *Let $\boldsymbol{A}$ be an $N \times N$ complex matrix, its singular value is bounded by*

$$\sigma(\boldsymbol{A}) \geq \frac{|\det(\boldsymbol{A})|}{2^{N/2-1} \|\boldsymbol{A}\|_{\mathrm{F}}}. \tag{30}$$

Proposition 6. *A lower bound and an upper bounds of $\mathcal{M}(\boldsymbol{s})$ in equation (18) are given by*

$$\frac{16}{9N^3} \leq \mathcal{M}(\boldsymbol{s}) \leq N^3 2^{N-4}.$$

Proof. Let us start from

$$\lambda_{\min}(\boldsymbol{R}^{-1})\boldsymbol{s}^T \boldsymbol{s} \leq \boldsymbol{s}^T \boldsymbol{R}^{-1} \boldsymbol{s} \leq \lambda_{\max}(\boldsymbol{R}^{-1})\boldsymbol{s}^T \boldsymbol{s}, \tag{31}$$

where $\lambda_{\min}(\boldsymbol{R}^{-1})$ and $\lambda_{\max}(\boldsymbol{R}^{-1})$ are the minimum and maximum eigenvalues of $\boldsymbol{R}^{-1}$, and $\boldsymbol{s}^T \boldsymbol{s} = N$. Moreover, given the fact that $\boldsymbol{R}$ and $\boldsymbol{R}^{-1}$ are positive definite (Lemma 2), we have $\lambda_{\min}(\boldsymbol{R}^{-1}) = 1/\lambda_{\max}(\boldsymbol{R})$ and $\lambda_{\max}(\boldsymbol{R}^{-1}) = 1/\lambda_{\min}(\boldsymbol{R})$. Eq. (31) can be rewritten as

$$N/\lambda_{\max}(\boldsymbol{R}) \leq \boldsymbol{s}^T \boldsymbol{R}^{-1} \boldsymbol{s} \leq N/\lambda_{\min}(\boldsymbol{R}). \tag{32}$$

The problem is then to bound the maximum and minimum eigenvalues of matrix $\boldsymbol{R}$.

First, let us focus on the upper bound of $\lambda_{\max}(\boldsymbol{R})$. Following the definition of matrix $\boldsymbol{R}$ in (16), we can factorize $\boldsymbol{R}$ as $\boldsymbol{R} = \boldsymbol{C}\boldsymbol{C}^T$, where $\boldsymbol{C}$ is given by

$$\boldsymbol{C} = \begin{bmatrix} \boldsymbol{J}_{N-1}\boldsymbol{s} & \boldsymbol{J}_{N-2}\boldsymbol{s} & \cdots & \boldsymbol{J}_2\boldsymbol{s} & \boldsymbol{J}_1\boldsymbol{s} & \boldsymbol{J}_{-1}\boldsymbol{s} & \boldsymbol{J}_{-2}\boldsymbol{s} & \cdots & \boldsymbol{J}_{2-N}\boldsymbol{s} & \boldsymbol{J}_{1-N}\boldsymbol{s} \end{bmatrix}, \tag{33}$$

That is, the columns of $\boldsymbol{C}$ are constructed by $\boldsymbol{J}_n\boldsymbol{s}$ for each integer $n \in [1-N, N-1]$ and $n \neq 0$.

In particular, we can bound the maximum singular value of C by (Lemma 4)

$$\sigma_{\max}(C) \leq \frac{3}{4}N^2. \tag{34}$$

Given (32) and (34), it follows directly that

$$\mathcal{M}(s) = s^T R^{-1} s \geq N/\lambda_{\max}(R) = N/\sigma_{\max}^2(C) \geq \frac{16}{9N^3}. \tag{35}$$

This gives us a lower bound for $\mathcal{M}(s)$.

Next, we bound the minimum eigenvalues of R.

The matrix C in (33) can further be partitioned as $C = [C_1 \ C_2]$, where C_1 is composed of N columns of C and C_2 contains the rest of the columns. In this partition, we require that the N columns in C_1 are mutually independent. This requirement can be easily met, for example, we could choose C_1 to be

$$C_1 = \begin{bmatrix} J_{N-1}s & J_{N-2}s & \cdots & J_{N-i}s & J_{-i}s & J_{-i-1}s & \cdots & J_{2-N}s & J_{1-N}s \end{bmatrix}, \tag{36}$$

where i can be $1, 2, ..., N - 1$.

Given $C = [C_1 \ C_2]$, we have $R = CC^T = C_1 C_1^T + C_2 C_2^T$.

Lemma 3 implies the following fact:

$$\lambda_{\min}(R) \geq \lambda_{\min}(C_1 C_1^T) + \lambda_{\min}(C_2 C_2^T) = \lambda_{\min}(C_1 C_1^T), \tag{37}$$

where the equality follows because $C_2 C_2^T$ is singular.

Then, the problem is to bound the minimum eigenvalue of $C_1 C_1^T$, or equivalently, the minimum singular value of C_1.

By Lemma 5, the minimum singular value of C_1 can be bounded by

$$\sigma_{\min}(C_1) \geq \frac{|\det(C_1)|}{2^{N/2-1} \|C_1\|_F}.$$

The construction in (36) suggests that the eigenvalues of C_1 can only be 1 or -1. In other words, $|\det(C_1)| = 1$ and

$$\sigma_{\min}(C_1) \geq \frac{|\det(C_1)|}{2^{N/2-1} \|C_1\|_F} = \frac{2^{2-N/2}}{N}. \tag{38}$$

Substituting (38) into (37), yielding

$$\lambda_{\min}(R) \geq \lambda_{\min}(C_1 C_1^T) = \sigma_{\min}^2(C_1) \geq \frac{2^{4-N}}{N^2}.$$

By (32), it follows directly that

$$\mathcal{M}(s) = s^T R^{-1} s \leq N^3 2^{N-4}. \tag{39}$$

$\square$

To verify the tightness of the derived bounds, we then exhaustive search to find $\inf \mathcal{M}(\boldsymbol{s})$ and $\sup \mathcal{M}(\boldsymbol{s})$ when $N \leq 30$. Simulation results indicate that the derived lower bound is close to $\inf \mathcal{M}(\boldsymbol{s})$, while the derived upper bound is far from tight. From the simulation, it seems that $\sup \mathcal{M}(\boldsymbol{s})$ is bounded by a constant value, rather than increasing as N increases. In the following, we conjecture a tight upper bound for $\mathcal{M}(\boldsymbol{s})$.

Conjecture. *For any phase code $\boldsymbol{s}$, a tight upper bound for $\mathcal{M}(\boldsymbol{s})$ is given by $\mathcal{M}(\boldsymbol{s}) \leq 37$. The equality is achieved by a Barker sequence of length* 13*, that is,*

$$\boldsymbol{s}_{\mathrm{B}} = [1, 1, 1, 1, 1, -1, -1, 1, 1, -1, 1, -1, 1]^{T}. \tag{40}$$

This conjectured bound is motivated from the study of the merit factor problem in the recent decades [28]. Until now, Baker sequence of length 13 is the best known sequence that yields the maximal merit factor [29]. For Barker sequence, a favorable property is that $|\boldsymbol{s}_{\mathrm{B}}^{T} \boldsymbol{J}_{\mathbf{0}} \boldsymbol{s}_{\mathrm{B}}| = N$, while $|\boldsymbol{s}_{\mathrm{B}}^{T} \boldsymbol{J}_{\boldsymbol{n}} \boldsymbol{s}_{\mathrm{B}}| = 1$ for $n \neq 0$. This is why we conjecture that a Barker sequence[11] can yield the maximal $\mathcal{M}(\boldsymbol{s})$.

REFERENCES

[1] OEIS, "The on-line encyclopedia of integer sequences," *avaliable: https://oeis.org/A000040*, 1964.

[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 and R. Padovani, "Implications of mobile cellular CDMA," *IEEE Commun. Mag.*, vol. 30, no. 12, pp. 38–41, 1992.

[4] M. I. Skolnik, *Radar handbook.* 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*, 2018.

[6] D. M. Burton, *Elementary number theory.* McGraw-Hill, 2005.

[7] R. S. Sutton and A. G. Barto, *Reinforcement learning: an introduction.* MIT press Cambridge, 1998, vol. 1, no. 1.

[8] M. L. Puterman, *Markov decision processes: discrete stochastic dynamic programming.* John Wiley & Sons, 2014.

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

[10] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

---

[11] Another famous conjecture is that there is no Barker sequence of length $N > 13$. Although not being proven, it had been shown that there is no Barker sequence of length $13 < N < 10^{22}$ [29].

[11] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of Go with deep neural networks and tree search," *Nature*, vol. 529, no. 7587, pp. 484–489, 2016.

[12] Y. Li, "Deep reinforcement learning: an overview," *arXiv preprint arXiv:1701.07274*, 2017.

[13] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *Nature*, vol. 550, no. 7676, p. 354, 2017.

[14] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A survey of monte carlo tree search methods," *IEEE Trans. Comput. Intell. AI*, vol. 4, no. 1, pp. 1–43, 2012.

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

[16] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel *et al.*, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *arXiv preprint arXiv:1712.01815*, 2017.

[17] K. S. Gilhousen, I. M. Jacobs, R. Padovani, A. J. Viterbi, L. A. Weaver, and C. E. Wheatley, "On the capacity of a cellular CDMA system," *IEEE Trans. Veh. Tech.*, vol. 40, no. 2, pp. 303–312, 1991.

[18] H.-H. Chen, *The next generation CDMA technologies*. John Wiley & Sons, 2007.

[19] C.-M. Chao, Y.-C. Tseng, and L.-C. Wang, "Reducing internal and external fragmentations of OVSF codes in WCDMA systems with multiple codes," *IEEE Trans. Wireless Commun.*, vol. 4, no. 4, pp. 1516–1526, 2005.

[20] D. V. Sarwate and M. B. Pursley, "Crosscorrelation properties of pseudorandom and related sequences," *Proc. IEEE*, vol. 68, no. 5, pp. 593–619, 1980.

[21] 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, 2001.

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

[23] J. Li, A. Huang, M. Guizani, and H.-H. Chen, "Inter-group complementary codes for interference-resistant CDMA wireless communications," *IEEE Trans. Wireless Commun.*, vol. 7, no. 1, 2008.

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

[25] R. M. Davis, R. L. Fante, and R. P. Perry, "Phase-coded waveforms for radar," *IEEE Trans. Aerospace and Electron. Syst.*, vol. 43, no. 1, 2007.

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

[27] M. Golay, "The merit factor of long low autocorrelation binary sequences (corresp.)," *IEEE Trans. Inf. Theory*, vol. 28, no. 3, pp. 543–549, 1982.

[28] T. Høholdt, "The merit factor problem for binary sequences," in *Int. Symp. Appl. Algebra, Algebraic Algorithms, and Error-Correcting Codes (AAECC)*. Springer, 2006, pp. 51–59.

[29] J. Jedwab, "What can be used instead of a barker sequence?" *Contemporary Math.*, vol. 461, pp. 153–178, 2008.

[30] 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, 1994.

[31] M. H. Ackroyd and F. Ghani, "Optimum mismatched filters for sidelobe suppression," *IEEE Trans. Aerospace and Electron. Syst.*, no. 2, pp. 214–218, 1973.

[32] Y. Shao, S. C. Liew, and T. Wang, "AlphaSeq: sequence discovery with deep reinforcement learning (source code)," *available: https://github.com/lintonshaw*, 2018.

[33] A. Knutson and T. Tao, "Honeycombs and sums of hermitian matrices," *Notices Amer. Math. Soc*, vol. 48, no. 2, 2001.

[34] V. Nikiforov, "Revisiting schur's bound on the largest singular value," *arXiv preprint math/0702722*, 2007.

[35] G. Piazza and T. Politi, "An upper bound for the condition number of a matrix in spectral norm," *J. Comput. Appl. Math.*, vol. 143, no. 1, pp. 141–144, 2002.