Decision-Time Planning with Monte-Carlo Tree Search

Johanna Hansen*
School of Computer Science
Mobile Robotics Lab (MRL)
McGill University
Montréal, QC
johanna.hansen@mail.mcgill.ca

Bobak Hamed-Baghi*
School of Computer Science
Mobile Robotics Lab (MRL)
McGill University
Montréal, QC
bobak.hamed-baghi@mail.mcgill.ca

Kyle Kastner*
Département d'Informatique et de Recherche Opérationnelle (DIRO)
Montreal Institute for Learning Algorithms (MILA)
Université de Montréal
Montréal, QC
kastnerkyle@gmail.com

Abstract

Planning against constraints is a crucial problem for automatic decision making. Limited decision time in the real world further compounds the challenge of choosing how to act in a given situation. We investigate variants of one promising approach to decision-time planning, Monte-Carlo Tree Search (MCTS), across three different domains and report results. Code and additional materials for this project are available at: **https://github.com/rllabmcgill/final-project-mcts_for_decision_time_planning**.

1 Introduction

In this report, we investigate one powerful anytime planning method, Monte-Carlo Tree Search (MCTS) [1], for solving sequential decision making tasks. Given an accurate representation of the future and unlimited time to compute, MCTS performs well, even when faced with large state or action spaces by *rolling out* many possible future scenarios to acquire an approximate (Monte Carlo) estimate of the value of taking a specific action from a particular state.

MCTS was introduced by Coulom in 2006 as a decision time method for choosing a promising action given a state. The recent success of AlphaGoZero [3] in the games of Go and AlphaZero [4] in Go, Shogi and Chess has inspired a bevy of other work (including our own) using Monte Carlo Tree Search such as [5], [6] [7]. In this paper we concentrate on our implementations of MCTS, for a full overview of MCTS and its many variants, please refer to [1]. We present experiments across three domains: Counterpoint Musical Exercises, Goal-Oriented Path Planning with Moving Obstacles, and OpenAI's implementation of Atari Ski (RAM) [8].

2 Experiments

In this section, we discuss implementation and results across several different environments. For the counterpoint experiment, we build a new multi-voice counterpoint environment in which an agent must select notes to harmonize against a *cantus firmus* without violating rules as described by

Prepared for the Winter 2018 course on Reinforcement Learning at McGill University (COMP 767)

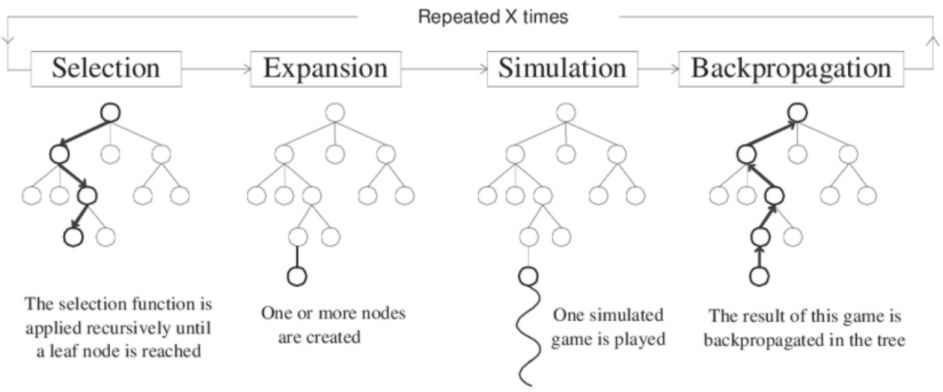


Figure 1: Monte-Carlo Tree Search

Figure 2: This figures (from [2]) demonstrates the iterative process by which Monte-Carlo Tree Search (MCTS) selects appealing actions at decision time.

The Study of Counterpoint [9]. The trajectory environment consists of a new custom environment in which an agent must navigate from a random start point to a goal without being hit by moving obstacles. In the Atari Ski environment, an agent must choose between three actions to ski down a hill as quickly as possible.

2.1 Counterpoint Exercises

One of the preeminent teaching tools for counterpoint composition dates back to 1725 during the Baroque era of classical music. *Gradus ad Parnassum* by Johann Joseph Fux [9] was an early formalization of "music theory" for counterpoint, which traces its roots to the earlier study of *punctus contra punctum* (note against note) in the 1300s. This instruction guide attempted to condense the methods used by many composers of the day into a set of teachable rules which slowly introduce the concepts of harmony and composition. Previous approaches [10] have analyzed the rules in this book, and defined a programmatic system for completing exercises in this style. Our approach relaxes the explicit probabilistic priors for each transition, favoring the more generic MCTS discovery procedure. This allows a more direct encoding of rewards, and suitable paths for maximizing these rewards are **discovered**, rather than prescribed. Music composition (and creative tasks in general) using algorithmic tools are widely studied, though we do not enumerate here [11, 12].

This work focuses on three voice, species one counterpoint. Three voice composition is particularly important to composition, as the three independent musical voices (soprano, tenor, and bass) potentially form the minimal building block of music, the harmonic triad. To quote Fux [9], "That three part composition is the most perfect of all is already evident from the fact that in it one can have a complete harmonic triad without adding another voice... to those who master three part composition the way to the composition of more parts is made quite easy".

Species one counterpoint is defined as "note against note", meaning each note choice takes one bar and each note is in reference to a fixed melodic line, known as a *cantus firmus* (**cf**) of equal tempo and note duration. All **cf** occur in the bass (bottom) voice in our procedure, with composed lines above in the soprano (top) and tenor (middle) voices though this is not a requirement of the methods used.

Applying MCTS to counterpoint requires a few primary components: a state space in which to make action choices based on the current state, an action space which defines the allowable actions from each state, a viable environment to check our "composition plan" for mistakes, and a reward function for any particular trace depending on success or failure.

Our state space consists of a fully visible trace, showing the entire cantus firmus (which we always place in the bass voice) to be planned against, as well as all previously chosen notes in the soprano and tenor. The cantus firmus was normalized against the last note (which is generally the tonic note, or the musical "root" of piece), which meant each cf was key invariant while roughly retaining the step-wise relationship between the current point and the "goal" (final note). This means a cf of

"C4, D4, E4, A3, D4, C4" would first translate to integer pitches of [60, 62, 64, 57, 62, 60], and subsequently be normalized relative to the final note, becoming [0, 2, 4, -3, 2, 0].

The two prediction voices (soprano and tenor) are defined relative to the cantus firmus, specifically as *intervals* relative to the cf. While negative intervals are possible in general (utilizing voice crossing), in this work we limit the intervals used to be strictly above the cf with no unity intervals, and each voice being relatively above the one below. This defines both a rough state representation, and an action space. We consider intervals from 0 to 28 pitch steps above the cf, where each pitch step represents a semi-tone in the musical scale. In this representation a two bar set of musical chords over three voices may be defined as [["E4", "C4", "A3",], ["G4", "E4", "C4"]]. This state would then reduce to the integer form of [[5, 3, -3], [5, 4, 0]].

At each step we additionally apply action space reductions which are dependent on several factors. The form of counterpoint used in this book is known as "modal" counterpoint, meaning each counterpoint should follow a particular "mode" of the musical scale. Discussion of modes and their musical meaning is beyond the scope of this report, but it suffices to say that the modal pruning *greatly* reduces the action space, only allowing intervals whose notes relative to the cf are within the chosen mode for the counterpoint exercise.

The final action pruning occurs relative to the previous states. One of the most fundamental principles of counterpoint is the idea of smooth voices, where the subsequent notes are nearby neighbors to the previous. We utilize a scheme whereby the plausible note actions are limited to only an interval of a perfect 4th (5 in our integer representation) above or below the previous note in a voice. We apply this limitation in both voices, but if *no* valid actions are found after these prunings, we gradually grow the allowed leap interval from a perfect 4th to a major 6th, with the major 6th being the largest allowable leap according to the instruction manual. If there are still no possible actions, the sequence is terminated and a negative reward given.

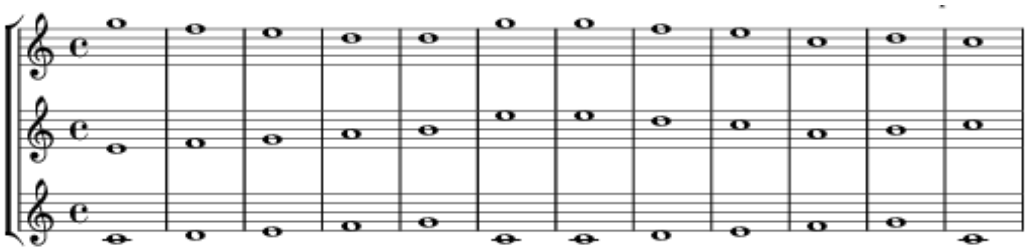


Figure 3: Agent trajectory (top two voices) composed against given *cantus firmus* (bottom voice)

The final chord of the counterpoint exercise has one additional action pruning applied, in order that the last voicing be "final" according to theory rules. This means the chord may only contain intervals of 3rds, 5ths, or 8ths, or octave relative versions of these intervals. This too is described in the instruction book, and forms the "punctuation" for the musical phrase being constructed.

Defining the environment for three voice, species 1 counterpoint requires additional constraints. We utilize a slightly simplified version of the true species 1 ruleset with three rules: bar consonance, next step, and parallel. The full ruleset includes checks for modality, which we choose to instead include in the action space pruning.

Bar consonance deals with the harmonic relationship between the chosen note set, in which we only allow harmonic relationships between each voice, in major, minor, and perfect flavors (M, m, P). The harmonic intervals in this work consisted of [m3, M3, P4, P5, m6, M6, P8] and their octave relations to a 2 octave offset [m10, M10, P11, P12, m13, M13, P15, m17, M17, P18, P19, m20, M20, P22, m24, M24]. This interval check is applied between each voice and every other. Intervals are defined using integer numbers representing the number of semi-tones in the interval, for example P5 being 7 or P8 being 12.

Next step rules deal with invalid interval leaps between consecutive notes in a voice, and are partially redundant with the action pruning described above. This rule largely prevents excessively large leaps, though it can also remove leap intervals disallowed in a particular style.

Parallel rules deal with arguably the most critical aspect of counterpoint, the relation of previous voice intervals to the next. As denoted in the manual [9] "Martini reduces the [interval rules] to one: progression forbidden is the direct motion into a perfect consonance". Briefly, this means that previous intervals of the perfect variety (denoted by the P prefix above) cannot move to following perfect prefixes. Failure to avoid "parallel fifths and octaves" is one of the most common beginner mistakes in counterpoint composition. We check each possible pair in order to eliminate any parallel perfect motion, in any voice.

Rewards are structured to further encourage smoothness in the top voice, which often carries the most melodic portion of musical phrases. We assign rewards which range from -1 to $-1 + \frac{n_s}{t_s}$ with n_s as number of valid steps and t_s as total number of cantus firmus steps, if a rule is violated. This results in a curriculum learning [13] reward where partial success is more valuable than immediate failure. If a trace proceeds through all steps without any mistake, the reward is calculated as $1 + \frac{1}{s}$, where s (smoothness) is $1 + \sum_{i,j} |t_i - t_j|$ and i and j are sequential neighbor pairs. Even without smoothness rewards on the middle voice, we see that it is often necessary to have the inner voice smooth, in order for the top voice to maximize overall reward. The presence of the cantus firmus means that the accompaniment rarely chooses the same note repeatedly, however without this cf a further diversity reward would likely be required.

To summarize our full environment procedure, take for example a 3 bar cantus firmus of ["C4", "D4", "C4"], which becomes [0, 2, 0] in integer representation. Our initial state will be [[], [], [0, 2, 0]], which is then fed to the MCTS procedure. After performing the normal MCTS child expansion/rollout procedure, checking the validity of rules each time, we may choose the action corresponding to [12, 7], a [P8, P5] interval combination. The state is then updated to [["C5"], ["G4"], ["C4", "D4", "C4"]], or [[12], [7], [0, 2, 0]]. We enter MCTS again, performing child expansion and rollouts, and perhaps choose [9, 3] ([M6, m3]), resulting in [["C5", "B4"], ["G4", "F4"], ["C4", "D4", "C4"]] or [[12, 9], [7, 3], [0, 2, 0]]. Lastly, we choose an ending chord consisting of [7, 4] ([P5, M3]) intervals, leading to [["C5", "B4", "G4"], ["G4", "F4", "E4"], ["C4, "D4", "C4"]]. Because the top voice is partially smooth, we receive a reward of $1 + \frac{1}{1+4}$.

The MCTS approach here uses PUCT [3], with a simple uniform prior over all valid action choices. UCT [14] is equally valid, however we choose to display results from PUCT for consistency purposes with other experiments, as well as future integrations with learned prior predictors, context based adaptation, and other modeling approaches.

2.2 Goal Oriented Navigation with Moving Obstacles

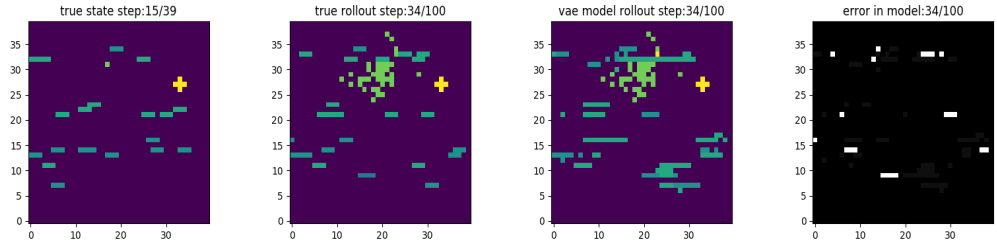
Navigation is an important part of mobile robotics applications, but it remains difficult in environments which are difficult to predict or observe. For this project, we wanted to develop a completely configurable, OpenAI Gym compatible, test environment from which to experiment in a world in which moving obstacles could be tuned to be more or less predictable.

Our navigation environment (depicted in Figure 4) consists of a fixed-size world of 40x40 cells with moving obstacles. In each episode, the agent and goal are initialized randomly. The agent has a limited number of steps (165 for a world size of 40x40) to reach the goal. At each step the agent must choose 1 of 8 possible angles to move in at a speed of 1 cell per step. Obstacles in this configuration of the environment can be thought of as cars driving on a highway. The cars enter the world frame from one side of the environment and exit the other side. For these experiments, the cars maintain a constant speed after they have entered the environment. The speed for a particular car is drawn from a normal distribution that is parameterized based on the class of the object. The class also dictates the obstacle's size and color. At any point in an episode, there tends to be more of the smaller, faster obstacles and fewer of the large, but slow obstacles, as described by a configurable Poisson distribution.

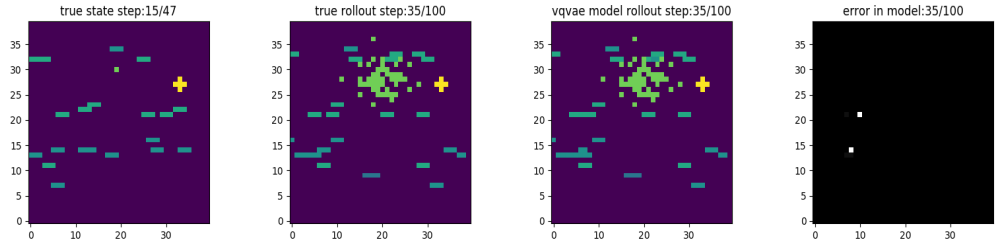
Navigation Reward Structure	
reach goal	$10 + \frac{10}{steps}$
collide	$-10 + \frac{-10}{steps}$
timeout	$\frac{-10}{steps}$

Table 1: Reward structure for trajectory agent.

Reward structure for our agent was important to the performance. We tried many reward structures including a simple $+1$ for reaching the goal and -1 for being hit by an obstacle. Ultimately, we found that it was important to incentivize short trajectories by including the number of steps taken by



(a) VAE State Representation



(b) VQ-VAE State Representation

Figure 4: This figure depicts our agent computing an optimal action for the 15th step of the game with 100 rollouts with a maximum rollout length of 100. In the top diagram, Figure 4a, the agent rolls out on an approximation of the true future state with a Variational AutoEncoder (VAE). The bottom agent, Figure 4b depicts an agent which computes best actions with an approximation of the true future state using a Vector Quantized-Variational AutoEncoder (VQ-VAE). The second column of both of these images is the true future state that the model in the third column is trying to represent. Model error is depicted in the fourth column where white pixels show false negatives (predicted free space where there was a car), gray pixels show false positives (predict car where there was free space), and black pixels depicting a correct estimate.

the agent and a relatively large bonus/penalty for important states. See Table 1 for a description of the final reward structure chosen for the experiments presented in this section.

We used a prior for node selection as described by the PUCT equation in [3]. Our experiments (see Table 2) showed that a simple prior which biased new leaf nodes in the relative direction of the goal improved performance over a uniform prior. This goal-oriented prior sets the 3 actions with angles closest to the goal to be $2.5\times$ as likely as the remaining actions. We found that setting this preference too high tended to cause the agent to be too greedy and step into obstacles on its way to the goal.

Initially our agent performed rollouts on a simulation of the true environment. This means that the MCTS had perfect information about the future states of the system. However, access to perfect future states is often not available in real-world tasks. Therefore, we are interested in building models of the world from which we can query instead of relying on the simulator. In time, we hope to develop a full forward model of the environment, however, for this project, we focused on developing and evaluating methods for compressed state representation. This minimal representation of the state will eventually be used to learn a dynamics model in future work.

We evaluated two forms of compressed state representation for the obstacles in the environment (not including the agent of the goal); a Variational AutoEncoder (VAE) [15] and a Vector Quantized Variational AutoEncoder (VQ-VAE) [16]. Our VAE and VQ-VAE each consisted consist of encoders and decoders with three convolutional layers each. In general, we found that VQ-VAE had more impressive reconstruction results (see Figure 4b). However, the this model uses a discrete nearest-neighbor embedding in the latent that we expect may be hard to use in future work. For rollouts over the noisier VAE, at each estimate of a state, we sample 20 times from the mu and $sigma$ of the latent space and conservatively consider the maximum estimated pixel value from all of the decoded estimates as our state representation.

We compare the priors and state representation over the same 100 episodes which were generated from distinct random seeds (unique initial position, goal position, and obstacles) and present the

Average Reward and (% of Games Won) over 100 Episodes			
Rollout Type	100 Rollouts, Limit 100	100 Rollouts, Limit 50	50 Rollouts, Limit 100
True state UP	-0.469 (46%)	-	-
True state GP	10.031 (99%)	8.875 (96%)	9.025 (94%)
VQ-VAE state GP	**10.232** (100%)	10.187 (100%)	9.430 (96%)
VAE state GP	9.212 (95%)	7.911 (91%)	8.003 (89%)

Table 2: Average episodic reward for each technique with the specified number of rollouts which were limited to the specified number of steps. Here UP stands for uniform prior and GP stands for goal-oriented prior. A higher reward indicates better performance (shorter path to goal).

Average Compute Time per Action Choice in Seconds over 100 Episodes			
Rollout State	100 Rollouts, Limit 100	100 Rollouts, Limit 50	50 Rollouts, Limit 100
True state UP	1.978	-	-
True state GP	3.84	2.11	0.72
VQ-VAE state GP	2.93	2.02	0.729
VAE state GP	2.76	1.72	0.56

Table 3: Average per-step compute time for each technique with the specified number of rollouts which were limited to the specified number of steps (where UP is uniform prior and GP is goal-oriented prior). Shorter compute time is not necessarily appealing as it often indicates that trajectories were cut short in the rollout by collision with an obstacle. This data does not include the time required to compute the state approximation, only the time required to perform a rollout and update the tree.

results in Tables 2 and 3. Perhaps surprisingly, the agent which uses VQ-VAE representation to perform rollouts actually slightly outperforms the agent planning on the true state. We suspect that this advantage may be due to our preference being set too high for the goal in the prior. Small mistakes in the VQ-VAE representation (as seen in the fourth column of Figure 4b) will cause the agent to behave more conservatively and reduce the likelihood of it making an eager dash for the goal and getting hit by an obstacle. In fact, the evidence seems to bear this out, if we only consider episodes in which the agent actually reached the goal (wasn't struck by an obstacle), the agent using the true environment for rollouts consistently achieves slightly shorter paths. Our preference for the goal in the prior seems to be too a bit too high. In our next iterations of the project, we would like to replace the hand-coded prior with a value net as seen in [3] so that it adjusts to the state. Lastly, we see from the table of results that the VAE performs quite a bit worse than both the true and VQ-VAE representation.

In Table 3 we present the average compute time per action for a number of different rollouts and rollout limits. With MCTS there is a tradeoff between the time allotted to compute and the optimality of the result. As one expects, performance generally improves as more compute time is allowed.

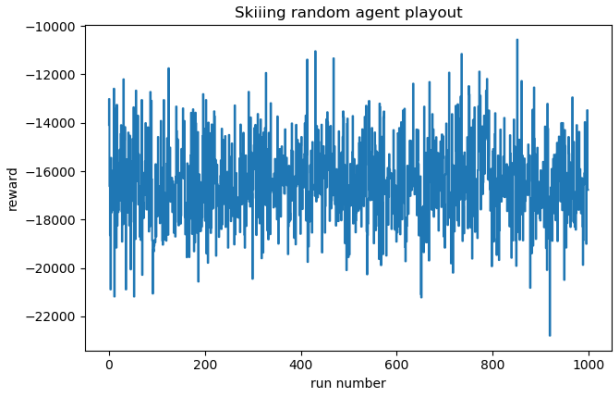
2.3 Atari Skiing (RAM)

In Atari Skiing, Players ski down a vertical screen with with various obstacles blocking their path and blue flagpoles indicating a target region where they should try to pass through. The goal of the game is to maximize the amount of flagpoles while simultaneously striving to achieve the best possible clearing time. Clearing time is not only a function of trajectory however, as hitting obstacles causes the skier to fall and thus wasting precious time and removing the forward momentum built up prior to the collision. Figure 5a shows a visual representation of the game for human players.

OpenAI gym provides a RAM based environment for the simulation of the Skiing game. The observation for this environment is the 128-Byte RAM of the simulator. The RAM version of this game was chosen despite the fact that a visual observation variant existed, as it was deemed simpler to train for a potential policy or value approximation scheme. The agent can select either of three deterministic actions from the action space: move-left, down, move-right. Move-left and move-right generate no momentum and thus rely on the 'down' action to generate momentum. This action space is constant (all three actions available at all times) and deterministic. The selected action at each step is applied for k frames, uniformly sampled from the set $2, 3, 4$. The rewards are structured as

penalty per step representing lost time, as well as negative reward if the end goal-post is missed. For reference, Figure 5b shows the performance of an agent acting upon a completely random policy in the environment. A better baseline exists if the agent simply applied the action 'down' at all times, yielding a mean reward of $R_{down} = -706$. By doing this the agent collides with only one obstacle during the episode.

(a) A screenshot of the Atari Skiing game.



(b) The performance of an agent with a uniformly random policy. The mean reward obtained over 1000 episodes is -16458

3 Conclusion

MCTS is a powerful tool for sequential decision making, given a highly accurate simulation of future environmental behavior. Our experiments allowed us to learn about MCTS and consider its applicability to our domains of interest. We developed intuition about tuning and performance across a variety of different sequential decision making problems and built up a codebase that will serve as a launching point to scale future research.

4 Group Contributions

Kyle Kastner completed the Counterpoint Exercises including the building of an extensive environment and implementing an MCTS agent. Johanna Hansen developed the trajectory environment and built an MCTS agent. Johanna, in consultation with Kyle, trained various neural networks with the goal of building a good state representation. Bobak Hamed-Baghi implemented MCTS for Atari Skiing and Tic-Tac-Toe.

References

[1] Browne, C., E. Powley. A survey of monte carlo tree search methods. *Intelligence and AI*, 4(1):1–49, 2012.

[2] Chaslot, G., S. Bakkes, I. Szita, et al. Monte-carlo tree search: A new framework for game ai., 2008.

[3] Silver, D., J. Schrittwieser, K. Simonyan, et al. Mastering the game of go without human knowledge. *Nature*, 550:354–, 2017.

[4] Silver, D., T. Hubert, J. Schrittwieser, et al. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815, 2017.

[5] H. S. Segler, M., M. Preuss, M. P. Waller. Learning to plan chemical syntheses. 2017.

[6] Anthony, T., Z. Tian, D. Barber. Thinking fast and slow with deep learning and tree search. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett, eds., *Advances in Neural Information Processing Systems 30*, pages 5360–5370. Curran Associates, Inc., 2017.

[7] Guez, A., T. Weber, I. Antonoglou, et al. Learning to search with mctsnets. *CoRR*, abs/1802.04697, 2018.

[8] Plappert, M., M. Andrychowicz, A. Ray, et al. Multi-goal reinforcement learning: Challenging robotics environments and request for research, 2018.

[9] Fux, J. J., A. Mann, J. Edmunds. *The study of counterpoint from Johann Joseph Fux's Gradus ad parnassum. Translated and edited by Alfred Mann, with the collaboration of John Edmunds*. W. W. Norton New York, rev. ed. edn., 1965.

[10] Farbood, M., B. Schoner. *Analysis and synthesis of Palestrina-style counterpoint using Markov chains*, vol. 2, pages 471–474. 2001.

[11] Briot, J.-P., G. Hadjeres, F. Pachet. Deep learning techniques for music generation - a survey. *CoRR*, abs/1709.01620, 2017.

[12] Browne, C. Towards MCTS for Creative Domains. In *Proc. Int. Conf. Comput. Creat.*, pages 96–101. Mexico City, Mexico, 2011.

[13] Bengio, Y., J. Louradour, R. Collobert, et al. Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, pages 41–48. ACM, New York, NY, USA, 2009.

[14] Kocsis, L., C. Szepesvári. Bandit based monte-carlo planning. In *In: ECML-06. Number 4212 in LNCS*, pages 282–293. Springer, 2006.

[15] Kingma, D. P., M. Welling. Auto-encoding variational bayes. *CoRR*, abs/1312.6114, 2013.

[16] van den Oord, A., O. Vinyals, k. kavukcuoglu. Neural discrete representation learning. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, R. Garnett, eds., *Advances in Neural Information Processing Systems 30*, pages 6306–6315. Curran Associates, Inc., 2017.