How Smart Do You Have To Be? Using Random Search To Solve Puzzles.

Robert Li

CMU-RI-TR-XX-XX
May 2020

The Robotics Institute
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Submitted in partial fulfillment of the requirements for the degree of Master of Science in Robotics.

Abstract

Physical puzzle solving is often seen as a difficult task due to the wide range of possible configurations of puzzles, and the need to reason about multiple abstract concepts. However, we would like to show that random search, with a little help from simple heuristics and optimization methods, is capable of solving these kinds of puzzles in a sample efficient manor. We use the PHYRE (PHYsical REasoning) benchmark, a set of simple mechanics puzzles in a 2D environment, to test our methods.

Acknowledgements

Contents

List of Figures

1. Introduction

The ability to solve novel problems with creative solutions has been a mark of intelligence [12]. Physical puzzles are a common type of problem used to test this ability in human children [5], and other animals such as members of the primate family [15], and members of the avian family [8]. As humans, we are capable of solving these puzzles with little prior knowledge of the solutions, using only our understanding of abstract concepts like kinematics, gravity, mass, friction, and inertia. This ability allows humans to easily solve a wide range of puzzles, an ability that artificial intelligence is currently lacking.

We wanted to explore problem solving with simple algorithms, which led us to use random search algorithms as a trial and error method to find reasonable starting points for gradient free optimization methods. The reasoning behind using random search is that as the complexity of a method grows, it becomes harder and harder to apply it to different scenarios. However, random search is simple and easy to implement for many different scenarios, and it can have good results with a little guidance. By using random search, we can solve a wide array of different puzzles, while maintaining sample efficiency.

Random search algorithms are a class of algorithms that uses some element of randomness or probability as part of their design. Common descriptors of random search algorithms include the term stochastic or the phrase Monte Carlo method. Random search algorithms have a wide array of uses, whether it's for path planning in the case of rapidly-exploring random trees (RRT) [10], or for solving ill-structured optimization problems [16]. Random search methods have seen recent success in model-free reinforcement learning [11], and in hyperparameter optimization [3].

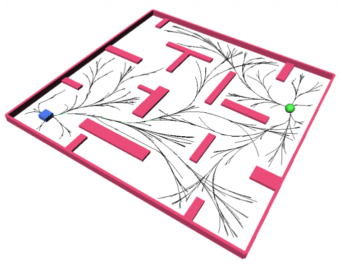


Figure 1.1: Example of RRTs solving a path planning problem [10]

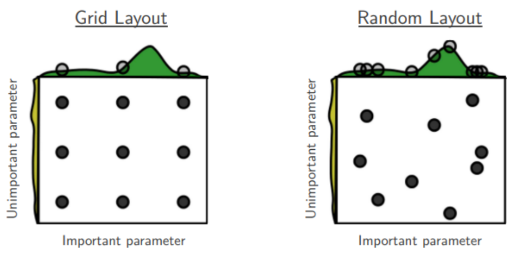


Figure 1.2: Example of Random Search used in hyperparameter optimization [3]

To evaluate our methods, we used the recently proposed PHYRE benchmark, which is a set of simple physics based puzzles in a 2D environment [1]. We chose this benchmark because all the tasks have the same goal, but there is a wide variety of puzzle configurations. In addition, the PHYRE benchmark has a strong emphasis on sample-efficiency, where each task allows only 100 trials. We will show that random search is capable of solving a significant number of puzzles within 100 trials.

PHYRE is a physical reasoning benchmark that consists of a set of physics puzzles in

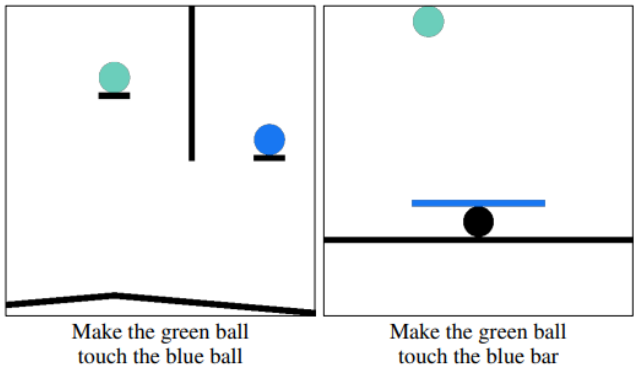


Figure 1.3: PHYRE Puzzles [1]

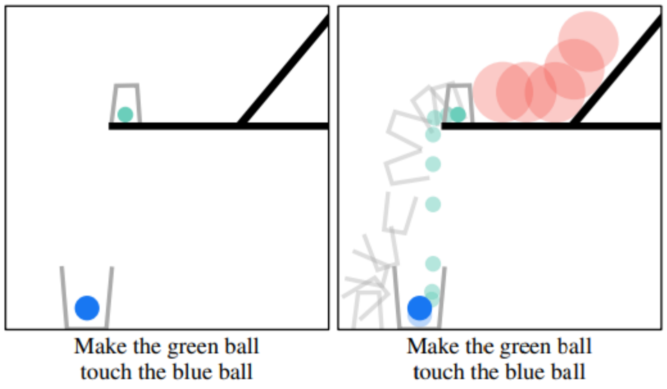


Figure 1.4: Solving a PHYRE Puzzle [1]

a simulated 2D environment. The puzzles all have the same goal state, which is to make two different colored objects touch each other for a specific period of time. A puzzle can be solved by placing one or more new objects in the environment before the simulation is started, so when the simulation is run, the goal state is reached. Examples of some of the puzzles in the PHYRE benchmark are in Figure 1.3, and an example of a solution to a puzzle is shown in Figure 1.4. We focused on solving the puzzles in the PHYRE-B set, which requires a single ball to be placed. As there are three parameters in the actions space, x position, y position, and ball radius, our search space is a 3D unit cube. In the PHYRE-B set, there are 25 different puzzles, and each puzzle has 100 different variations.

The three main goals of the PHYRE benchmark revolve around physical reasoning, generalization, and sample-efficiency. We will show that random search is capable of solving puzzles with basic physical reasoning, that random search can generalize between different tasks, and that random search is sample efficient.

We will explain our work in applying random search and optimization methods to the PHYRE benchmark. As the puzzles in the PHYRE benchmark are designed to be difficult for a random search to solve, we introduce two simple heuristics to help. More specifically, the first heuristic attempts to move the two objects closer by placing the action ball to collide with one of the two objects, based on their starting locations, to provide a force to move the two objects closer. As the first heuristic only takes the starting positions into account, our second heuristic uses the same idea as the first heuristic, but instead of taking just the starting positions into account, it takes into account the entire sequence of positions.

In addition to adding heuristics to random search, we also apply optimizations methods as a second step. We apply optimizations methods in order to reduce the number of attempts that the random search needs to make. We will show the results of applying both Bayesian and Random Optimization as a second step

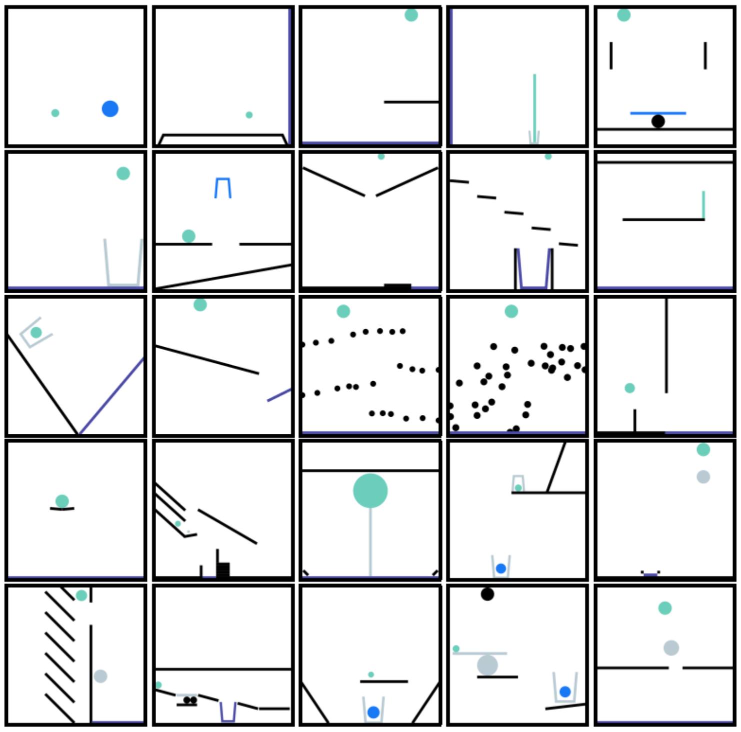


Figure 1.5: Puzzles in the PHYRE-B Set

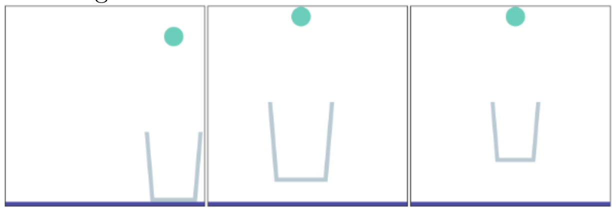


Figure 1.6: Examples of variations of a PHYRE puzzle

2. Related Work

Previous work on using random search to solve complicated tasks includes path planning, where Rapidly-exploring Random Trees (RRTs) were shown to be an effective method in solving single-query path planning problems in high-dimensional configurations spaces. In a manner similar to classical bidirectional search, RRT-Connect [9] grows two RRTs from the start and goal positions. As the trees grow, they explore the space around them, and advance towards each other with a greedy heuristic, with the goal of connecting the two trees.

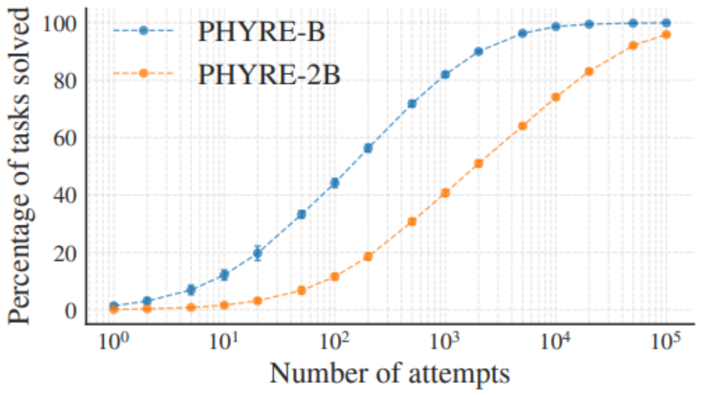
Random search has also been shown to be capable of beating manual and grid search strategies for hyper-parameter optimization [3]. In the case of grid search compared to random search, the success rate of grid search depends greatly on the spacing of the grid, and the actual shape of the surface, as shown in Figure 1.2. In addition, if we know that the optimal solution takes up some percent x of the search space, then we can calculate the number of trials that random search needs to find a solution with a high probability. From equation 2.1, if 5% of the space is optimal, then if we want a a 95% chance of finding an optimal solution with random search, we only need ≈ 60 trials.

$$1.0 - (1.0 - x)^n > y \tag{2.1}$$

In more recent research, random search has been used in reinforcement learning, where various methods known as evolution strategies [13] (ES) have shown to be competitive with more complicated algorithms such as DQN and A3C [14]. Evolution Strategies are a type of black box optimization algorithms inspired by natural evolution. At every iteration, a set of parameter vectors are modified with some random noise, and then their objection function values are used to determine the highest scoring vectors. These vectors are then recombined together to create the next set of parameters for the next iteration. At the same time, an even simpler algorithm called Augmented Random Search (ARS)[11] has also been shown to be comparable to other state-of-the-art methods.

3. Random Search With Heuristics

We will first discuss how we used random search to solve the puzzles. From the examples of random search compared to grid search for hyper-parameter optimization, it seems like random search would solve a significant number of puzzles given 100 attempts. However, the PHYRE puzzles were designed so that completely random search would be unable to solve the majority of the puzzles within 100 trials. As we can see from Figure 3.1, only 45% of the tasks can be solved by random search within 100 trials.



(a) Percentage of tasks solved by a *random* agent (y-axis) as a function of the number of attempts (x-axis; log scale) for both PHYRE tiers.

Figure 3.1: Image from [1]

3.1 Simple Heuristic

In order to narrow the search space that random search had to explore, we came up with a simple heuristic to guide our search. The simple heuristic places the action ball on either side of the two objects, with the goal of letting the action ball collide with either the blue or red objects, to provide a sideways force to move them closer to each other. Figure 3.2 and Figure 3.3 show two examples of solutions that use different objects for the same puzzle. Figure 3.4 shows a visualization of all the actions that the simple heuristics considers plotted on top of each other. Figure 3.5 shows a visualization of all the actions that the simple heuristic considers plotted in the actions space.

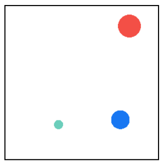


Figure 3.2: Example of a solution that uses the blue object

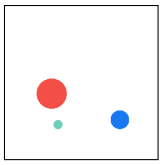


Figure 3.3: Example of a solution that uses the green object

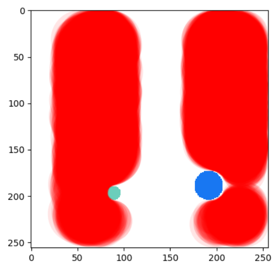


Figure 3.4: Visualization of actions that the simple heuristic considers in the scene

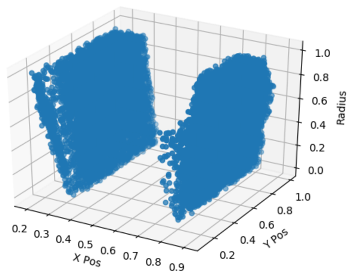


Figure 3.5: Visualization of actions that the simple heuristic considers in the action space

3.1.1 Results of Simple Heuristic

Before going into the results of the simple heuristic with random search, we need to first explain how PHYRE sets up its tests. Given the 25 puzzles, and 100 different variations of each puzzle, there are 2500 total different puzzles. PHYRE splits these puzzles into 10 groups, each with a training, validation, and test set. To be consistent with the benchmark, even though we do not use any training, we evaluated our results on only the test set. In addition to the groups, PHYRE also uses two different templates, Cross and Within, where in the within template, the same types of puzzles are in the test and training set, while in the cross template, different types of puzzles are in the test compared to the training set.

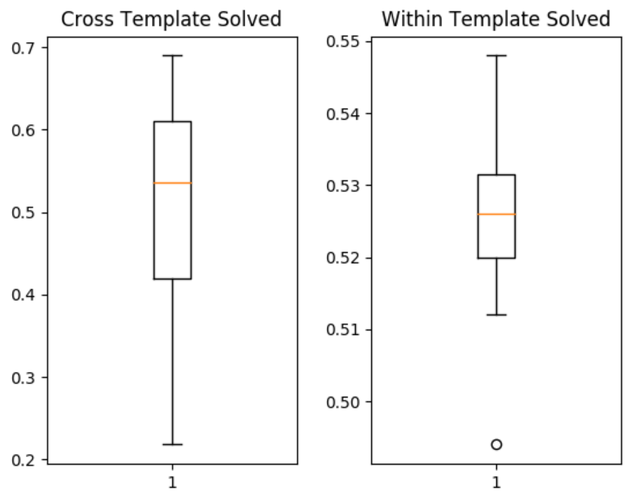


Figure 3.6: Percentages of Test Tasks Solved Per Group (1-10) with the simple heuristic. Note that the Y-axes have different scales in the two graphs.

From Figure 3.6, we can see that in the cross template, the simple heuristic can solve on average 50% of the tasks across all groups. In the within template, the simple heuristic can solve an average of 52% of the tasks across all groups. In comparison to completely random search, the simple heuristic allows random search to solve 5% more tasks on average.

3.2 Sequential Heuristic

Although the simple heuristic showed some improvement over completely random search, it was a small improvement compared to the performance of completely random search. The main issue with the simple heuristic is that it only took into account the beginning of the puzzle, instead of the entire sequence of motion that the objects go through. In Figure 3.7 and in Figure 3.8 we can see that the actions the simple heuristic considers do not solve this particular puzzle.

To rectify this issue, we introduce a sequential heuristic that considers the sequence of positions that the blue and green objects in the puzzle go through. It then considers placing a ball in the path of the blue and green objects, accounting for gravity as well. This heuristic also does not take into account the relative positions of the green and blue objects, as we wanted to increase the size of the search space to include more solutions. In Figure 3.9 and in Figure 3.10 we can see that the actions the sequential heuristic considers many more actions compared to the simple heuristic.

3.2.1 Results of Sequential Heuristic

From Figure 3.11, we can see that in the cross template, the sequential heuristic can solve on average 62% of the tasks across all groups. In the within template, the sequential heuristic

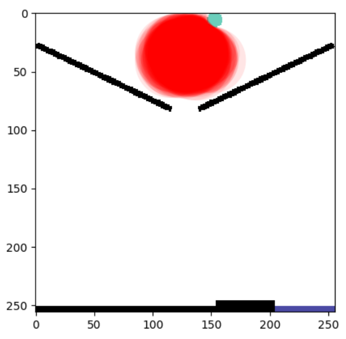


Figure 3.7: Visualization of actions that the simple heuristic considers in the scene

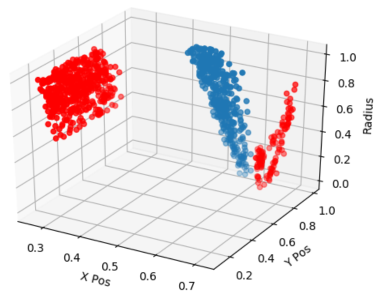


Figure 3.8: Visualization of actions that the simple heuristic considers in the action space along with solutions. Blue points are actions that the simple heuristic considers, and red points are actions that solve the puzzle in Figure 3.7

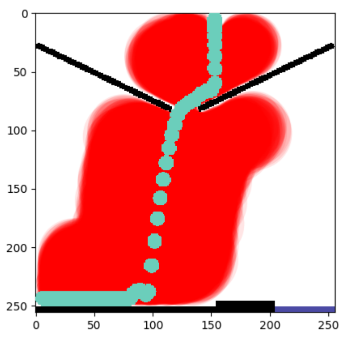


Figure 3.9: Visualization of actions that the sequential heuristic considers in the scene

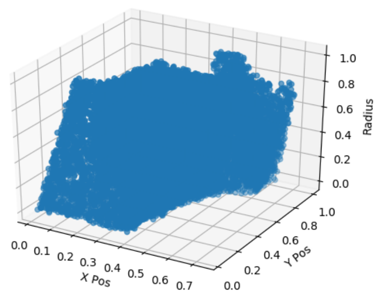


Figure 3.10: Visualization of actions that the sequential heuristic considers in the action space along with solutions.

can solve an average of 66% of the tasks across all groups. In comparison to the simple heuristic, the sequential heuristic allows random search to solve 10% more tasks on average.

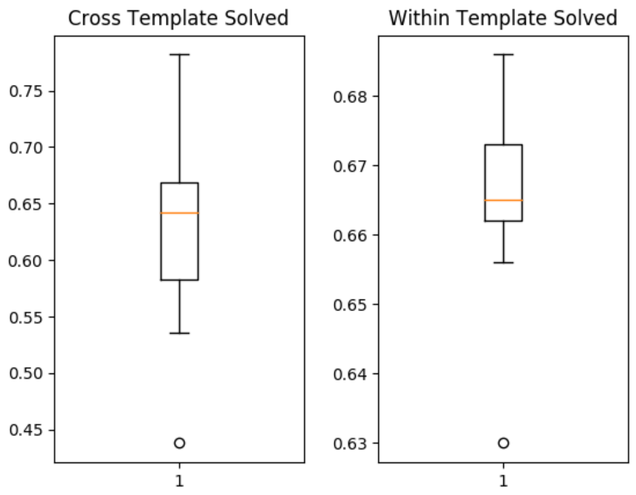


Figure 3.11: Percentages of Test Tasks Solved Per Group (1-10) with the Sequential heuristic

4. Local Optimization

Next, we will cover how we used optimization methods to improve the results of random search. The reasoning behind using optimization after random search comes from the idea that if the random search comes close enough to a solution, then local optimization should be able to refine the parameters that random search came up with into a solution. By relaxing the condition that random search has to find an exact solution, we are effectively increasing the size of the solution space.

4.1 Cost Function

In order to use optimization methods on the PHYRE puzzles, we first needed to come up with a cost function, as we only have success and failure states from the puzzles. As our cost function, we combine the number of frames that the blue and green objects spend in contact with each other, scaled from 0 to 1, and the distance between the green and blue objects at the end of the sequence, also scaled from 0 to 1. The reason why we used these two different measures is that the contact number encodes the goal, while the distance smoothed out the objective function.

4.2 Optimization methods

In addition to the cost function, we also needed black box optimization methods, as we do not have an analytical form of the objective function that we can differentiate. The two methods we used are Bayesian optimization and Augmented Random Search (ARS)[11]. Bayesian optimization uses a surrogate probability model of the objective function that is meant to be easier to optimize than the actual objective function. After the optimal parameters of the surrogate model are found, then the actual objective function is evaluated using the found parameters, and the surrogate model is updated. These steps repeat until either the maximum number of iterations or some other stopping conditions are reached. We used the Python package Hyperopt [2], which uses Tree of Parzen Estimators (TPE) as the surrogate model [4]. Augmented Random Search is a method that is similar to hill climbing, where the algorithm calculates a finite difference of the cost of a set of random directions sampled around the current set of parameters, and then takes a step in the average direction, weighted by the finite difference of the costs. We used both Bayesian optimization and ARS after random search finds a set of parameters that reached a cost above a handpicked threshold.

4.3 Results

4.3.1 Results of Bayesian Optimization

From Figure 4.1, we can see that in the cross template, the sequential heuristic can solve on average 66% of the tasks across all groups. In the within template, the sequential heuristic

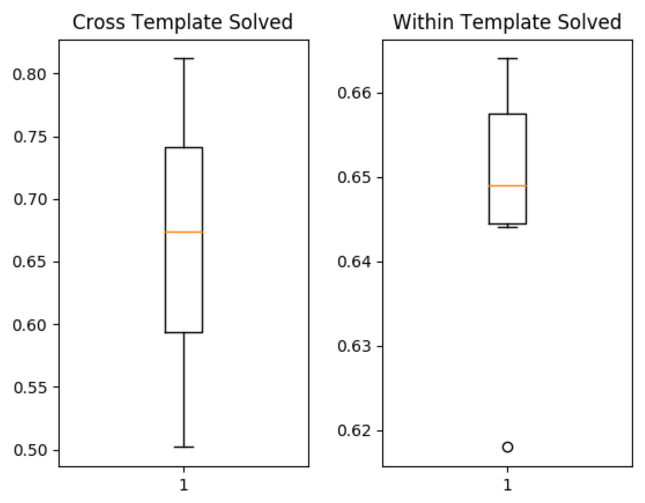


Figure 4.1: Percentages of Test Tasks Solved Per Group (1-10) with the Sequential heuristic and Bayesian Optimization

can solve an average of 66% of the tasks across all groups. In comparison to using only the sequential heuristic, adding Bayesian Optimization allows us to solve 4% more tasks on average.

4.3.2 Results of Augmented Random Search

From Figure 4.2, we can see that in the cross template, the sequential heuristic can solve on average 71% of the tasks across all groups. In the within template, the sequential heuristic can solve an average of 82% of the tasks across all groups. In comparison to using only the sequential heuristic, adding Bayesian Optimization allows us to solve 10% to 15% more tasks on average.

What is notable about this result is that ARS is capable of solving more tasks compared to Bayesian optimization. Our intuition is that our cost function, depending on the puzzle, is not one that Bayesian optimization can model easily. In addition, we found that Bayesian optimization spent more time exploiting points that were close to a solution, but not an actual solution, due to the fact that Bayesian Optimization does not take the binary success or failure states into account. In comparison, ARS will always try to explore new regions, instead of exploiting previously explored regions.

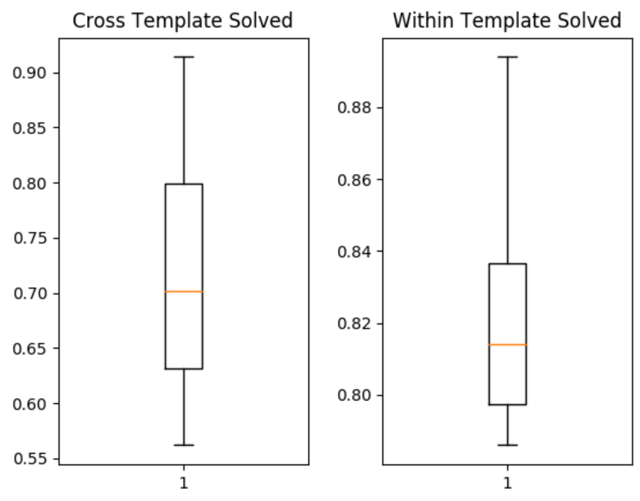
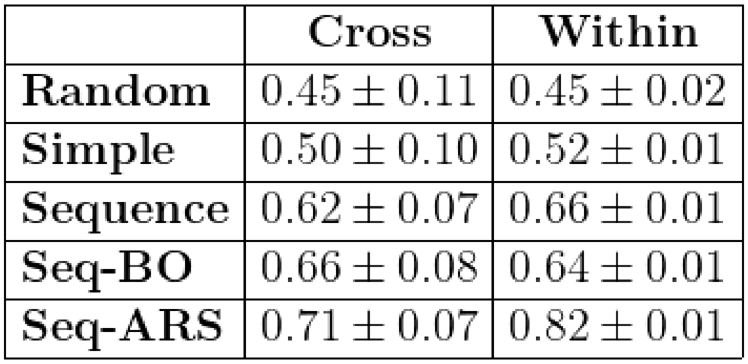


Figure 4.2: Percentages of Test Tasks Solved Per Group (1-10) with the Sequential heuristic and Augmented Random Search

5. Overall Results



	Cross	**Within**
Random	0.45 ± 0.11	0.45 ± 0.02
Simple	0.50 ± 0.10	0.52 ± 0.01
Sequence	0.62 ± 0.07	0.66 ± 0.01
Seq-BO	0.66 ± 0.08	0.64 ± 0.01
Seq-ARS	0.71 ± 0.07	0.82 ± 0.01

Figure 5.1: Average percent of tasks solved solved over all groups with one standard deviation

From 5.1, we achieved significant improvements over completely random search. On average, random search using the sequential heuristic and ARS can solve 26% to 37% more tasks compared to using just random search.

6. Conclusions and Future Work

We have shown that using random search augmented with simple heuristics and optimization methods is capable of solving non trivial physics puzzles in a limited number of trials. We have outlined how random search can be easily improved by adding simple heuristics, and how optimization methods can further improve random search.

Our methods do have some obvious limitations and drawbacks. Firstly, we are only capable of solving the PHYRE-B puzzles, as the heuristics used would not apply to many of the PHYRE-2B puzzles that require some forms of chain reactions, as we do not use any understanding of how an action changes the motion of the objects in the scene. In addition, as solving one puzzle is independent from solving another puzzle, we are unable to leverage past trials from puzzles of the same type, or similar puzzles.

Some immediate future work could address those limitations, by adding some future predictions to allow for chain reactions. Some examples of using future predictions includes work from qualitative physics, which has been used to solve mini-golf like puzzles, reasoning about the motion of a ball in a qualitative manner [7]. Another example of leveraging future predictions comes from learning visual models, which have been used to play billiards [6].

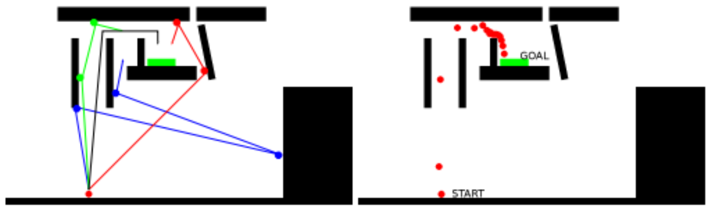


Figure 6.1: An example of Qualitative Physics solving a mini-golf puzzle [7]

In addition, the PHYRE benchmark is a simulated environment, it would be interesting apply our approach on real life problems with real robots. One possible problem could revolve around grasping and manipulation problems, selecting exactly how an object should be held for a specific task. Although we are using more and more complicated methods to solve problems in AI and robotics, I think it is important to remember that simple methods are still capable with only a little tweaking.

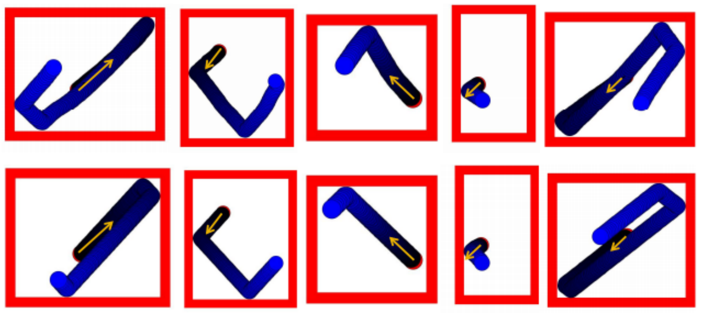


Figure 6.2: An example of predicting the motion of a billiards ball. The images on the top are the predicted motions, and the images on the bottom are the ground truth trajectories [6]

Bibliography

[1] Anton Bakhtin, Laurens van der Maaten, Justin Johnson, Laura Gustafson, and Ross Girshick. Phyre: A new benchmark for physical reasoning, 2019. URL `https://arxiv.org/abs/1908.05656`.

[2] J. Bergstra, D. Yamins, and D. D. Cox. Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. In *Proceedings of the 30th International Conference on International Conference on Machine Learning - Volume 28*, ICML'13, page I–115–I–123. JMLR.org, 2013.

[3] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *J. Mach. Learn. Res.*, 13:281–305, February 2012. ISSN 1532-4435.

[4] James Bergstra, Rémi Bardenet, Yoshua Bengio, and Balázs Kégl. Algorithms for hyper-parameter optimization. In *Proceedings of the 24th International Conference on Neural Information Processing Systems*, NIPS'11, page 2546–2554, Red Hook, NY, USA, 2011. Curran Associates Inc. ISBN 9781618395993.

[5] Lucy G. Cheke, Elsa Loissel, and Nicola S. Clayton. How do children solve aesop's fable? *PLOS ONE*, 7(7):1–12, 07 2012. doi: 10.1371/journal.pone.0040574. URL `https://doi.org/10.1371/journal.pone.0040574`.

[6] Katerina Fragkiadaki, Pulkit Agrawal, Sergey Levine, and Jitendra Malik. Learning visual predictive models of physics for playing billiards, 2015. URL `https://arxiv.org/abs/1511.07404`.

[7] Xiaoyu Ge, Jae Hee Lee, Jochen Renz, and Peng Zhang. Hole in one: Using qualitative reasoning for solving hard physical puzzle problems. In *ECAI*, 2016.

[8] Sarah A. Jelbert, Alex H. Taylor, Lucy G. Cheke, Nicola S. Clayton, and Russell D. Gray. Using the Aesop's Fable Paradigm to Investigate Causal Understanding of Water Displacement by New Caledonian Crows. *PLOS ONE*, 9(3):1–9, 03 2014. doi: 10.1371/journal.pone.0092895. URL `https://doi.org/10.1371/journal.pone.0092895`.

[9] J. J. Kuffner and S. M. LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, volume 2, pages 995–1001 vol.2, 2000.

[10] Steven M. LaValle and Jr. James J. Kuffner. Randomized kinodynamic planning. *The International Journal of Robotics Research*, 20(5):378–400, 2001. doi: 10.1177/02783640122067453. URL `https://doi.org/10.1177/02783640122067453`.

[11] Horia Mania, Aurelia Guy, and Benjamin Recht. Simple random search provides a competitive approach to reinforcement learning, 2018. URL `https://arxiv.org/abs/1803.07055`.

[12] Richard E. Mayer. *Human Nonadversary Problem Solving*, pages 39–56. Springer US, Boston, MA, 1989. ISBN 978-1-4684-8015-3.

[13] Ingo. Rechenberg. *Evolution strategy; Optimization of technical systems according to principles of biological evolution. With an afterword by Manfred Eigen.* Frommann-Holzboog [Stuttgart-Bad Cannstatt], 1973. ISBN 3772803733.

[14] Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. Evolution strategies as a scalable alternative to reinforcement learning, 2017. URL `https://arxiv.org/abs/1703.03864`.

[15] Elisabetta Visalberghi and Luca Limongelli. Acting and understanding: Tool use revisited through the minds of capuchin monkeys. In A. Russon, Kim A. Bard, and S. Parkers, editors, *Reaching Into Thought: The Minds of the Great Apes*, pages 57–79. Cambridge University Press, 1996.

[16] Zelda B. Zabinsky. *Random Search Algorithms.* American Cancer Society, 2011. ISBN 9780470400531. doi: 10.1002/9780470400531.eorms0704. URL `https://onlinelibrary.wiley.com/doi/abs/10.1002/9780470400531.eorms0704`.