A Comparison of Tree Search Methods for Graph Topology Design Problems

Ada-Rhodes Short, Bryony L. DuPont, and Matthew I. Campbell
Oregon State University, USA

In this paper, we discuss the relevance and effectiveness of two common methods for searching decision trees that represent design problems. When design problems are encoded in decision trees they are often multimodal, capture a range of complexity in valid solutions, and have distinguishable internal locations. We propose the use of a simple Color Graph problem to represent these characteristics. The two methods evaluated are a genetic algorithm and a Monte Carlo tree search. Using the Color Graph problem, it is demonstrated that a genetic algorithm can perform exceptionally well on such unbounded and opaque design decision trees and that Monte Carlo tree searches are ineffective. Insights from this experiment are used to draw conclusions about the nature of design problems stored in decision trees and the need for new methods to search such trees and lead us to believe that exploitative methods are more effective than rigorously explorative methods.

1. Introduction

AI tree-searches are a common method for the creation of generative designs. This requires the problem to first be represented as a decision tree. Problems represented as decision trees have a benefit over conventional numerical optimization because the design space can have arbitrary complexity as opposed to being limited to a fixed vector of decision variables as in optimization.

Therefore, in this paper, we are exploring tree-search methods for finding the best solution to a design problem. It is our conjecture that the design decision trees we define in this paper are different from typical tree-search spaces explored by computer scientists. For example, the majority

Design Computing and Cognition DCC'18. J.S. Gero (ed),
pp. 81-97. © Springer 2018

of tree-search problems can be defined as path-planning problems or game trees. Path-planning trees are distinguished by the fact that the tree terminates at easily discernible goal states and the tree often contains monotonicities in approaching that goal. Game trees also terminate in goal or end-game states despite the fact that two or more agents control the decision making and are typically operating under counterproductive utility functions.

Design decision trees are marked by four unique qualities. First, they are often *multimodal*. There is rarely a monotonicity in the metrics that can be used to guide us towards a solution. Second, the solutions are of *unbounded* complexity meaning that non-terminal states in the tree can be a valid solution even though additional decisions can be made on them to make more complex solutions. As result, there is no clear sense of a valid goal state, and in some cases, even the starting seed may be seen as a valid solution. Third, because design decision trees are often comprised of a structure (in this paper we use a graph structure), the design has *locations* within it that can be leveraged in the subsequent decision making. For example, if a generative graph grammar [1] is used to define transitions in the decision tree, then the mapping of the left-hand-side of grammar rules through the recognition process may be reasoned about with some independence from the application or change produced by the rule (as indicated by the right-hand-side of the rule). Finally, design problems often contain states that are not evaluable. This means that the effect of each decision cannot be readily determined if the resulting state does not update the defined metric of quality. As a result, sometimes multiple cascading decisions are required to arrive at an evaluable state. For example, the comfort, dynamic response, or aesthetics of a bicycle cannot be determined without completing decisions on all relevant parts such as the drive-train, frame, wheels, and suspension. Sometimes predictions can be made but within an automated process, the effort to make such predictions would be significant over merely invoking a few more decisions to arrive at an evaluable state. We refer to this final quality as *opaqueness*.

Typically, deterministic methods like best first search (e.g. A* [2]) are used in path-planning algorithms, but these are rarely useful for the generic class of “design” trees that are described here. G for searching trees – also known as Genet[3] – has stagnated in recent decades in favor of the more generic concept of Monte-Carlo Tree Search (MCTS) methods[4]–[6]. However, the capabilities of these methods for problems portraying the four characteristics above (multimodality, unbounded, location, and opaqueness) is not well studied. This paper is a first attempt at exploring these methods to understand which are most applicable to design problems represented as decision trees.

1.1 Aims

This paper aims to establish an easily evaluable test problem to explore how different tree search methods perform as applied to graph topology design problems. In this paper, an emphasis was placed on the clear description of the implementation and methods for the purpose of repeatability, and to lay a foundation for future work that uses other methods and approaches to study the Color Graph design problem.

2. Significance

This work has profound potential significance in the field of automated design for three reasons. First, it describes a class of problem that is rarely studied in the abstract. Second, it establishes a standard evaluation metric for unbounded opaque topological design problems. Finally, it presents findings that are widely generalizable to many real-world automated design problems.

2.1 Problem Definition

The unbounded opaque design decision tree problem has three unique qualities. They are 1) multimodal and non-monotonic, 2) unbounded, and 3) contain multiple internal locations.

Multimodality complicates the design and analysis of a system. This is because the design cannot be evaluated until it is complete, as early choices may be a good in the beginning but lead to complications later on. An example of this would be constructing a multistory building to minimize cost. If a choice is judged by its value before the building is completed, then it will likely not have a foundation can sufficiently support later levels. However, if the whole building is designed first and then evaluated, the designer will be able to determine if the foundation was sufficient.

The arbitrary complexity of the unbounded opaque design decision tree creates further complications. Traditional tree search methods are capable of finding an optimal solution in a bounded tree, but because this class of design decision tree can have a potentially infinite number of choices the problem become intractably large and no state can be described as globally optimal.

Multiple internal locations make the problem more complex. A new location is added the system as it is constructed, resulting in a factorial branch factor. This means that the problem can become intractably large very quickly, and it becomes impossible to significantly sample the space.

2.2 Potential Applications and Generalization

While the exploration of unbounded opaque design decision trees is academically interesting for their own sake, this work is broadly generalizable to many applications. Including Automated Metallic-Organic Framework [7] design (directly inspired this work), architectural design [8], mechanical structures [9]–[12], piping systems like HVAC [13], and truss design [14]–[16]. It should be noted that Color Graph is capable of being feasibly represented as a binary GA which may not be true for all design problems, however insights gained should still be generally applicable.

3. Method

In this paper, we present the results of two methods applied to the same design problem. A simple design problem was created that can be evaluated in a very short amount of time, roughly 0.001243 seconds on the machine that was used for this experiment.

3.1 The Color Graph Design Problem

A Color Graph is a directed graph composed of a seed node, n_o, to which red, orange, yellow, green, blue, or violet colored nodes are added. In addition to the seed node, colored nodes can be added to other colored nodes already existing in the graph. Fig. 1 shows three examples of Color Graph candidates found in the search tree.

Fig. 1 Three examples of a Color Graph

The design decision tree for a Color Graph alternates between location decisions and color decisions. There are six branches coming off the root representing the six potential colors for the added node. From each of those six color options, there are two branches representing potential locations in the Color Graph to add the next node. One for the Color Graph seed node, n_o, and one for the first node added, n_1. From each of those location options the design decision tree branches again with the six color options. The design decision tree continues to repeat this way between color and location, with the number of location options increasing every

time a node is added to the Color Graph. Fig. 2 shows the design decision tree and corresponding Color Graph for the first four nodes added.

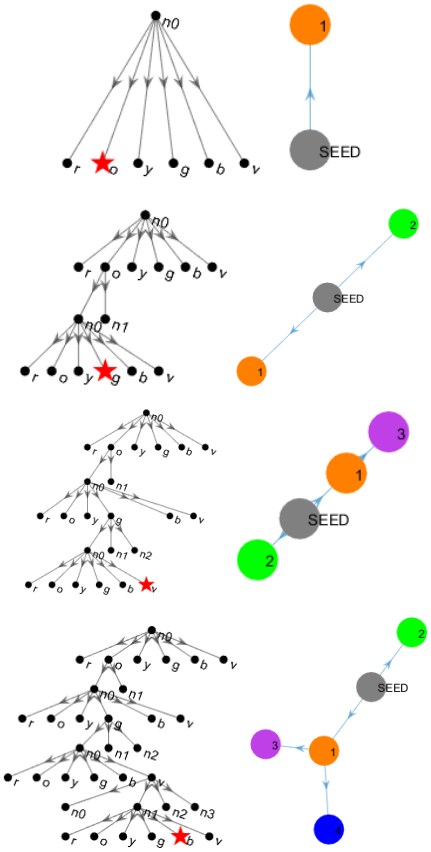


Fig. 2 Design Decision Tree and Corresponding Color Graph

The edges of a Color Graph design determine its quality. The edge's scores are determined by the source node and target node of each edge. For the case study presented in this paper, the edges have three arbitrary properties: α, β, and γ, with a range of -5 to 5. These edge properties are independent and are randomly generated. When a completed Color Graph is evaluated, the scores for each edge are summed giving a three-dimensional score $[\Sigma\alpha, \Sigma\beta, \Sigma\gamma]$. Then, the design is rated on its proximity to a target score, which was $[0,0,0]$ for this paper. Table 1 shows example edge scores for a Color Graph, and an example Color Graph is scored based on its edges in Figure 3.

Table 1 Edge Properties

		Target Node																	
		α						β						γ					
		Red	Orange	Yellow	Green	Blue	Violet	Red	Orange	Yellow	Green	Blue	Violet	Red	Orange	Yellow	Green	Blue	Violet
Source Node	Red	-3.15	3.83	-0.71	3.18	3.27	-3.49	0.38	1.06	-1.94	1.85	-2.20	-0.37	-0.06	-2.46	2.57	-3.85	-3.62	-2.53
	Orange	3.56	3.45	-4.28	4.09	1.92	-1.10	1.55	-1.42	0.54	-0.96	4.06	-2.65	-2.82	2.11	0.74	2.83	4.89	1.81
	Yellow	3.72	-4.54	2.93	-3.12	-4.44	0.85	0.18	3.95	-4.32	-2.88	-3.94	-3.00	1.88	1.33	-0.24	-4.92	0.98	-3.91
	Green	1.14	0.76	1.13	-0.04	-3.27	1.43	-2.37	0.66	0.54	2.31	-1.89	4.99	0.74	2.62	1.65	2.80	1.22	1.29
	Blue	0.30	-4.27	-1.79	2.81	1.42	-4.48	4.56	-2.19	-4.55	1.45	2.52	-3.87	4.15	-3.46	1.03	4.29	3.66	4.23
	Violet	0.93	-3.20	-4.42	3.99	4.58	4.47	-0.40	-4.58	-0.39	-0.71	-0.64	1.59	-3.41	0.96	4.89	-1.42	-4.40	-0.16
	Seed	-1.51	-0.22	-0.64	-3.32	-4.89	-2.38	1.34	-3.98	3.83	0.47	1.08	4.99	0.09	-4.52	3.58	0.86	3.58	-2.48

Figure 3 shows an example of how a Color Graph is scored using the edge properties from **Error! Reference source not found.**. The edge between n_0 (the seed node) and n_1 (the orange node) has an α value of-0.22, a β value of -3.98, and a γ value of -4.52. We sum these with the edge scores from the remaining four edges and get totals of $\alpha = -2.72$, $\beta = -2.10$, and $\gamma = 3.04$ or as a three dimensional coordinate in a design space [-2.72,-2.10,3.01]. We compare this to our target design score of [0,0,0] and determine the quality of the design by its proximity to the target. This can be found by calculating the distance between the two points using Euclidean distance, giving the design a final quality score of 4.59. A perfect score would be a 0 meaning that the design perfectly recreated the target design.

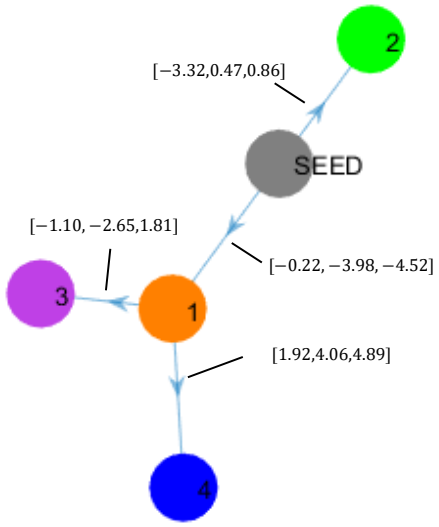


Figure 3 Color Graph Edge Properties

The Color Graph problem exhibits all of the properties of interest of the design decision tree. The design objective function for evaluating solution quality is multimodal and non-monotonically related to the number of

nodes present in the Color Graph. For example, if the Color Graph in Figure 3 was only the first four nodes of a larger design and a designer added a red node to n_1, then the design quality would improve, but if a designer added a violet node to n_1 the quality of the design would decrease. This problem is compounded by the opaqueness of the design decision tree. The Color Graph design process is opaque as the final design performance cannot be determined from the performance of an incomplete Color Graph design. In many real-world cases, opaqueness exists in designs that are not immediately or intuitively obvious to a human, due to a large number of sensitive design variables and multimodality of the design. The Color Graph design decision tree is unbounded because it has no set limit on the number of nodes that can be added to the Color Graph. Lastly, it possesses an increasing number of internal locations, as the locations where nodes can be placed increases with the number of nodes already present in the Color Graph. This gives the design decision tree for the Color Graph a branching factor of $n \times 6$, making its growth both geometric and factorial. Fig. 4 shows a sub-section of the Color Graph design decision tree with 9 levels (selecting node color 5 times and node location 4 times).

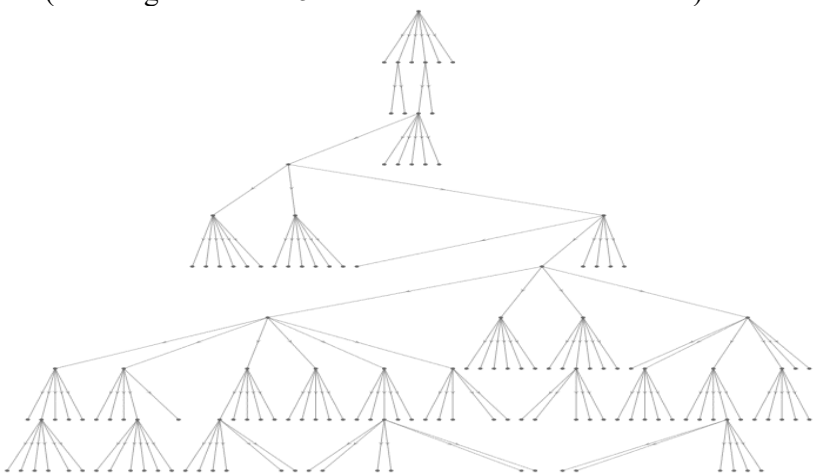


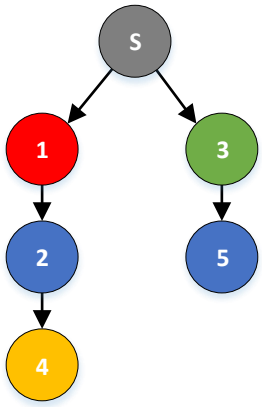
Fig. 4 A section of the design tree for a Color Graph

3.2 Algorithms evaluated

Two algorithms were evaluated on their ability to design a Color Graph. The chosen algorithms were a genetic algorithm [17] and a Monte Carlo tree search [4]. We implemented both methods in the 2017 edition of MATLAB [18].

3.3 Genetic Algorithm

A Genetic Algorithm (GA) is a metaheuristic design algorithm inspired by natural selection. A GA is one of a larger class of bioinspired algorithms called Evolutionary Algorithms (EA). For the Color Graph problem, the individual graph designs are represented by a set of chromosomes representing the location where a node is added, and the color of the added node. For example a chromosome could be [red, seed; blue, node-1; green, seed; yellow, node-2; blue, node-3] as shown in Fig. 5, additionally, we can say the length of the set of chromosomes is itself defined by an additional chromosome that was not varied during these trials. The effect of modulating the length though can be explored in future work.



Node	Color Added	Location Added
1	Red	S
2	Blue	1
3	Green	S
4	Yellow	2
5	Blue	3

Fig. 5 Example Color Graph with chromosome [red, seed; blue, node-1; green, seed; yellow, node-2; blue, node-3]

The algorithm starts by randomly generating 100 parent solutions. The algorithm then rates each solution with a fitness function, in this case, the Color Graph edge evaluation. The 10 top-performing individuals are copied into the next generation. The remaining 90 spots in the next generation are filled with the offspring of two parents randomly selected from the previous generation. Next, there is a small probability that a node color will randomly mutate. The GA is performed for 20 generations, and the final design is recorded.

The GA's greatest advantage is not its ability to search a tree in the traditional sense, but instead, they generate a constantly improving set of completed solutions. This has two advantages: 1) GAs find completed solutions, and 2) they do not have to follow a traditional search path and can make large moves to completely new branches. One potential weakness of

the GA is that while it can quickly find a good solution, it is highly stochastic and there is no guarantee that the GA will converge on a globally optimal solution. Genetic algorithms have a tendency to become fixated on a region and never explore beyond it unless a serendipitous mutation should arise to introduce new regions of improvement. So, one could prescribe a level of quality that is considered good enough and running the GA until the level of quality is reached.

Each of the 100 solutions in all 20 generations was evaluated and had its design and the quality score recorded in a cell array. When a new generation was created, the top 10 offspring in the previous generation were copied into the new generation. Next, a normal distribution with $\sigma = 8$ and $\mu = 0$ was used in a Cumulative Density Function (CDF) to select two parents. Two random values between 0 and 1 were generated and used to look up the inverse value in the normal CDF. These numbers would correspond to the rank of the two parents. The chromosomes of the offspring were randomly selected from the two parents, with equal frequency. Next, the random mutation would occur. For each added node, there was a probability of 0.05 that it would randomly mutate to a different color. This would preserve the overall structure of the Color Graph while switching the design decision tree to a different, but similar branch. The best scoring design was recorded into the final results, along with the number of generations needed to reach the design. The GA algorithm is shown below in Fig. 6.

Algorithm 1: GeneticAlgorithm

Input: Number of nodes N, and edge properties α, β, and γ, target properties t, generation size b, and number of generations G.

Output: A color graph design

1. Randomly generate b offspring
2. Compute offspring's α, β, γ properties
3. Compute offspring on proximity to t
   Loop Process
4. **for** $g = 2 : G$ do
5. Clone top 10 offspring to new generation
   Loop Process
6. **for** offspring $11 : b$
7. Select random parents biased to stop scoring of last generation
8. Select chromosomes randomly from parents
9. Randomly mutate chromosomes

Fig. 6 The Genetic Algorithm

3.4 Monte Carlo Tree Search Algorithm

A Monte Carlo Tree Search (MCTS) is a type of heuristic search algorithm that is often used in the analysis of games. MCTS consists of four basic steps: 1) selection, 2) expansion, 3) simulation, and 4) back-propagation. During selection, MCTS uses a policy to select the best option currently available. A policy is a set of rules developed by MCTS that dictate what decisions should be made. Starting from the root node, the seed in a Color Graph, the search traverses until a leaf is reached in the decision tree. When a leaf is reached, MCTS begins expansion. During expansion, MCTS adds the next round of potential choices to the branch. MCTS then performs simulation, consisting of selecting a leaf node that was just added and randomly sampling the potential branches beneath it until completed designs have been generated. The completed designs are then scored on quality. Finally, quality scores are back-propagated through the branch and back to the root, expanding on the previous policy by adding one more level of informed decision. This is performed for all the adjacent leaves at this level.

One reason for interest in MCTS is that it has been successfully applied to similar problems trees in game theory before [19], however, when applied to problems that could be classified as unbounded opaque design decision trees, MCTS has been shown to have inferior performance. A motivation for studying and publishing this work is to better understand and describe why MCTS underperforms on unbounded opaque design decision trees, such as Color Graph.

The best implementation found to store the quality scores of nodes was inside a digraph object (a graph with directed edges) in MATLAB directly [20]. Starting at the root location (the Color Graph seed), the design decision tree was expanded to add the six possible color choices. A sample of 100 random designs was taken. The sample size of 100 was selected after performing a parameter sweep on sample sizes to determine what generated a policy the most effectively. To compare MCTS more directly to the GA, parallelization of the search was not used. This was to create a baseline for comparison showed the general feasibility of the method, not the computer's ability to brute force the problem. In order to keep MCTS from running for an infeasibly long time, a time was implemented that stops MCTS after 20 minutes has elapsed and records the current best quality score and policy. The MCTS algorithm is shown below in Fig. 7.

Algorithm 2: Monte Carlo Tree Search

Input: Number of nodes N, and edge properties α, β, and γ, target properties t, sample size s.

Output: A color graph design

INITIALIZATION
Select the root node
SELECTION
Loop Process
While design policy has not reached length n
  While design policy exists
    Select branch with the best score
  End while
  EXPAND
  If terminal leaf is a location in the color graph
    Add six new leaves representing color options
  Elseif terminal leaf is a color decision
    Add a number of leaves equal to the number of nodes in the color graph
  End if
  SIMULATION
    Loop Process
  While new unexplored leaf exists
    Generate s random completed design branches
    Calculate the average score for decision tree leaf
  End while
  BACK PROPAGATION
  While current node $\neq$ root
    Select node back one level in the branch
    Compute the average score for that node based on *SIMULATION*

Fig. 7 Monte Carlo Tree Search Algorithm

3.5 Experimental Setup

For both the GA and MCTS method, we conducted an experiment in which Color Graphs with 3, 5, and 10 added nodes are designed. Each al-

gorithm explores graphs of variable sizes because the unbounded nature of the design decision tree means that more choice could always be made, therefore it is important to study how each method behaves as the size of the decision tree grows.

For each method and size of Color Graph 10 trials were performed. 10 sets of random edge properties were generated prior to performance of the experiment, one for each trial. This was to ensure that the results would not be biased due to of a single set of properties, and enables comparison across methods and graph sizes.

The metrics of interest are the quality score of the designed graph, the length of time needed to reach the solution, and the number of graph designs analyzed during the process. Additionally, we recorded the number of generations before the solution was discovered (for the GA), the depth of the policy (MCTS). This allowed us to gain additional insight on the capability of the methods, even if they fail to generate a completed design within the allotted 20-minute limit. The best Color Graph from each trial is stored in a cell array for later review.

In addition to the GA- and MCTS-generated Color Graphs, purely random Color Graphs (consisting of randomly chosen colors placed in random locations) will be generated for each trial as a control, and analyzed as a baseline for comparison.

4. Results

The entire experiment took approximately 7 hours and 45 minutes to run on an ordinary desktop computer with a 3.4 GHz Intel Xeon CPU [21] and 16 GB of RAM. A summary of the mean algorithm scores (with 0 being the best possible score) is shown in Fig. 8.

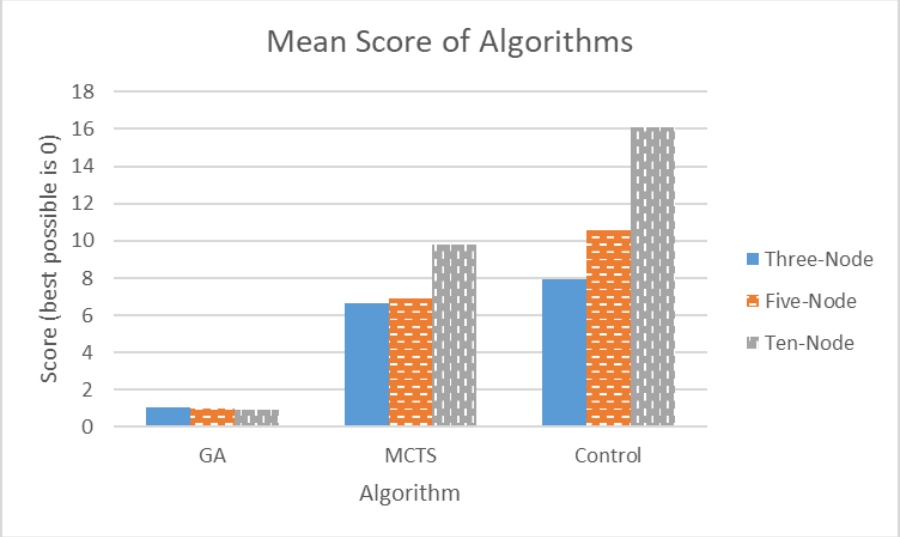


Fig. 8 Summary of Mean Scores

4.1 Three-Node Color Graph

In the first test, each method was used to generate a Color Graph with three added nodes, and we performed 10 trials for each of the methods. The methods were scored on their ability to improve Color Graph quality, with a best possible score of 0. Table 2 shows a summary of the results.

Table 2 Results of the Three-Node Test

	Three-Node Color Graph						
	Genetic Algorithm			Monte Carlos Tree Search			Control
	Quality Score	Run Time (s)	Nodes Evaluated	Quality Score	Run Time (s)	Nodes Evaluated	Quality Score
Mean	1.044	7.46	2000	6.630	309	6951	7.945
std	0.457			2.45			0.711

The best performing method was the GA with a mean score of 1.044, and a standard deviation of 0.457. MCTS had a mean score of 6.6296 and a standard deviation of 2.453. For this size, MCTS was only marginally better than the control and was 1.84 standard deviations away from a purely random design. The purely random control had a mean score of 7.945 with a standard deviation of 0.712.

4.2 Five-Node Color Graph

Again, the best performing method was the GA with a mean score of 0.9843, and a standard deviation of 0.413. MCTS had a mean score of 6.9008 and a standard deviation of 3.89. This significantly out-performed the purely random control, with a mean score of 10.59 and standard deviation of 0.929, but still performed much worse than the GA. Table 3 shows a summary of the results.

Table 3 Results of the Five-Node Test

	Five-Node Color Graph						
	Genetic Algorithm			Monte Carlos Tree Search			Control
	Quality Score	Run Time (s)	Nodes Evaluated	Quality Score	Run Time (s)	Nodes Evaluated	Quality Score
Mean	0.9843	11.6	2000	6.901	1221	17700	10.59
std	0.413			3.88			0.929

Another notable result is that for the five-node Color Graph, only three of the 10 MCTS trials successfully developed a full five-step design decision tree policy during the allotted 20 minutes. The other seven trials were only able to develop a four-step design decision tree policy, meaning that the final node's location and color are indeterminate and the final score was based on expected value if the branch were to be randomly completed. For comparison, the run that took the longest only lasted 11.6 seconds.

4.3 Ten-Node Color Graph

As in the first two tests, the best performing method was the GA with a mean score of 0.9290, and a standard deviation of 0.393. MCTS had a mean score of 9.774 and a standard deviation of 5.54, which is worse than

Table 4 Results of the Ten-Node Test

	Ten-Node Color Graph						
	Genetic Algorithm			Monte Carlos Tree Search			Control
	Quality Score	Run Time (s)	Nodes Evaluated	Quality Score	Run Time (s)	Nodes Evaluated	Quality Score
Mean	0.9290	21.3	2000	9.774	1217	17880	16.07
std	0.393			5.54			1.29

During the 10-node test, MCTS completely failed to generate a completed policy within the allotted 20 minutes. In four of the trials, MCTS developed a five-node design decision tree policy, in five trials it developed a four-node design decision tree policy, and in one trial it only developed a three-node design decision tree policy. The longest the GA took to complete was 21.3 seconds.

4.4 Discussion of Results

Analysis of the results leads to several interesting observations.

The first and most obvious observation is that the GA significantly outperformed MCTS in both quality score and runtime with a p-value of 0.0001 or less in all three cases. This was likely related to the branching factor of the design decision tree being $n \times 6$ resulting in a tree that rapidly becomes too large to feasibly search. MCTS struggles with this because it must search through the actual tree in order to find good design solutions. The GA avoids this problem by not relying on the design decision tree structure, and instead, working directly on designs.

It is expected that this observation is broadly generalizable outside of these two methods. For example other Evolutionary Algorithms (EA) that work directly on the Color Graph without considering the design decision tree would likely be able to find a solution relatively quickly, on the other hand, algorithms that search through the design decision tree directly, such as an Ant Colony Optimization (ACO), are likely to fail for find particularly good solutions.

A second notable result is the usefulness of MCTS was very sensitive to the size of the Color Graph. For the three-node Color Graph, MCTS did not significantly perform better than random (p-value of 0.1205), because it was not able to find a particularly good node due to how large the design decision tree becomes. However, as for the five-node Color Graph test, MCTS performs significantly better than random (p-value of 0.0091). This appears to be because MCTS is capable of finding an okay solution, but as the problem grows larger pure random selection performs increasingly poorly. However, the capability of MCTS is limited as the problem size continues to grow because the runtime needed quickly becomes infeasible.

5. Conclusion

Based on the results of the experiment, it can be concluded that a Genetic Algorithm (GA) is much better suited than a Monte Carlo Tree Search (MCTS) to the problem of designing systems with design decision trees

that are multimodal, unbounded, and contain multiple internal locations. In future work we hope to explore this further and validate that it is a result of how the methods utilize the design decision tree different.

Additionally, it has been shown that the Color Graph design problem serves as a good benchmark for the study and comparison of various forms of automated design methods in real-world unbounded problems. The designs of Color Graphs can be evaluated at a rate of approximately 1000 graphs per second on commonly available desktop computers while preserving characteristic multimodality, unbounded complexity, and multiple internal locations.

5.1 Future Work

Future work will focus on further exploration of the Color Graph design problem using a wider variety of methods. Additionally, experimentation will be performed to explore questions of option design complexity, the complexity of design properties and behavior, and the development of new and novel methods for automated system design.

6. Acknowledgments

This material is based upon work supported by the National Science Foundation under grant CMMI-1662731. Any opinions, findings, and conclusions or recommendations presented in this paper are those of the authors and do not necessarily reflect the views of the National Science Foundation.

7. References

[1] G. Rozenberg and H. Ehrig, “Handbook of Graph Grammars and Computing by Graph Transformation", 1997.

[2] L. N. Kanal and V. Kumar, *Search in artificial intelligence*. Springer-Verlag, 1988.

[3] J. R. Koza, Genetic programming: on the programming of computers by means of natural selection. The MIT press, 1992.

[4] C. Browne and E. Powley, “A survey of monte carlo tree search methods,” *IEEE Trans. Intell. AI Games*, vol. 4, no. 1, pp. 1–49, 2012.

[5] D. Perez *et al.*, “Solving the Physical Traveling Salesman Problem: Tree Search and Macro Actions,” *IEEE Trans. Comput. Intell. AI Games*, vol. 6, no. 1, pp. 31–45, Mar. 2014.

[6] D. Perez, P. Rohlfshagen, and S. M. Lucas, “Monte Carlo Tree Search: Long-term versus short-term planning,” in *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, 2012, pp. 219–226.

[7] C. A. Manion, R. Arlitt, I. Y. Tumer, M. I. Campbell, and P. A. Greaney, "TOWARDS AUTOMATED DESIGN OF MECHANICALLY FUNCTIONAL MOLECULES," in *Volume 2A: 41st Design Automation Conference*, 2015, p. V02AT03A004.
[8] H. Koning and J. Eizenberg, "The language of the prairie: Frank Lloyd Wright's prairie houses," *Environ. Plan. B Plan. Des.*, vol. 8, no. 3, pp. 295–323, 1981.
[9] J. Patel and M. I. Campbell, "An Approach to Automate Concept Generation of Sheet Metal Parts Based on Manufacturing Operations," in *Volume 1: 34th Design Automation Conference, Parts A and B*, 2008, vol. DETC2008-4, pp. 133–142.
[10] J. Patel and M. I. Campbell, "Topological and Parametric Tune and Prune Synthesis of Sheet Metal Parts Compared to Genetic Algorithm," in *AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*, 2008.
[11] A. Swantner and M. I. Campbell, "Topological and parametric optimization of gear trains," *Eng. Optim.*, vol. in review, pp. 1–18, Mar. 2012.
[12] P. Radhakrishnan and M. I. Campbell, "A graph grammar based scheme for generating and evaluating planar mechanisms," in *Design Computing and Cognition '10*, 2010, pp. 663–679.
[13] W. R. J. Patterson and M. I. Campbell, "PipeSynth: An Algorithm for Automated Topological and Parametric Design and Optimization of Pipe Networks," in *ASME Conference Proceedings*, 2011, vol. 2011, no. 54822, pp. 13–23.
[14] A. Hooshmand and M. I. Campbell, "Truss layout design and optimization using a generative synthesis approach," *Comput. Struct.*, vol. 163, pp. 1–28, Jan. 2016.
[15] K. Shea, E. Fest, and I. F. C. Smith, "Developing intelligent tensegrity structures with stochastic search," *Adv. Eng. Inform.*, vol. 16, no. 1, pp. 21–40, 2002.
[16] P. Shankar, J. Ju, J. D. Summers, and J. C. Ziegert, "DETC2010- DESIGN OF SINUSOIDAL AUXETIC STRUCTURES FOR HIGH SHEAR," *Eng. Conf.*, pp. 1–10, 2010.
[17] D. Whitley, "A genetic algorithm tutorial," *Stat. Comput.*, vol. 4, no. 2, pp. 65–85, 1994.
[18] "MATLAB - The Language of Technical Computing," 09-Dec-2015. [Online]. Available: http://www.mathworks.com/products/matlab/. [Accessed: 09-Dec-2015].
[19] C. B. Browne *et al.*, "A survey of monte carlo tree search methods," *IEEE Trans. Comput. Intell. AI Games*, vol. 4, no. 1, pp. 1–43, 2012.
[20] "Graph with directed edges - MATLAB." .
[21] "Intel® Xeon® Processor E3-1240 v2 (8M Cache, 3.40 GHz) Product Specifications," *Intel® ARK (Product Specs)*. [Online]. Available: https://ark.intel.com/products/65730/Intel-Xeon-Processor-E3-1240-v2-8M-Cache-3_40-GHz. [Accessed: 16-Dec-2017].