Combining EA and MCTS for improving optimization algorithms

Marina Díaz Uzquiano[1], Denis Pallez, and Jean-Paul Comet

Université Côte d'Azur, Sophia Antipolis 06410, France

Keywords: Evolutionary Algorithms · Monte Carlo Tree Search · RL-based EA

1 Introduction

With an increasing interest in the world of Artificial Intelligence, improving optimization algorithms for better performance has become a topic of interest. Finding better approaches to optimization or search algorithms could signify getting better, more-accurate solutions with less computation, and therefore having algorithms that can easily scale to realistic tasks. Integrating two different areas in AI for this purpose is an approach that has been looked into [7], since one could make up for the deficiencies of the other.

The objective of this research project is to find in the literature hybrid approaches in Reinforcement Learning and Evolutionary Computation, in particular, combining Monte Carlo Tree Search, and Evolutionary Algorithms. In order to later propose new possible approaches to combine the two techniques and potentially use the result in a bioinformatics setting.

1.1 Monte Carlo Tree Search

Machine Learning (ML) is a discipline in computer science that consists on the analysis and development of algorithms and techniques that allow machines to improve their performance on a problem-solving task by using data without being explicitly programmed to do so.

Reinforcement Learning (RL) is a ML technique that focuses on observing a space of possible solutions, taking actions, and evaluating them. This evaluation is a way of "rewarding" the best actions taken, and the goal of RL is maximising this reward.

Monte Carlo Tree Search (MCTS) is an RL algorithm, in particular, it is a heuristic tree search algorithm. What tree search algorithms do is to systematically search on a tree data structure. This data structure consists of a hierarchical set of nodes connected between them without loops. The term "heuristic" makes reference to the strategy of finding solutions to problems without guaranteeing finding the optimal one; but providing, nonetheless, a good-enough solution given the resources at hand.

In order to reach optimal solutions, during MCTS, the search tree is grown by doing random samples of the search space. MCTS starts at the root of the tree and iterates over four different steps as shown in figure (1): selection, expansion, simulation, and propagation [2]. The algorithm is usually run until a final state is reached or a computational budget is attained.

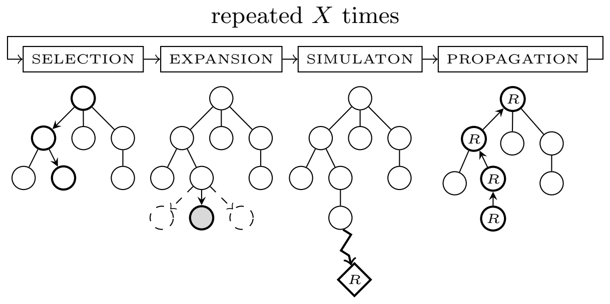

Fig. 1. Schematic representation of MCTS

Selection This step consists on selecting which node to expand in the next step. The algorithm tries to choose the node that is more urgent to expand, this is, that has unvisited children and that is not a terminal node. This decision-making step faces the dilemma of whether to exploit areas that we know can have some potential to find an optimal solution, or to explore our search space and look in areas that have not yet been sampled, and maybe reach a new and more optimal solution. A formula for the selection step that does well at facing the exploration-exploitation problem is UCB (Upper Confidence Bound) because it accounts for the times a node is visited (exploration) and for the times every child node of the parent node is visited (exploitation). For MCTS the Upper Confidence Bounds equation is known as UCT for Upper Confidence bounds for Trees.

Other approaches are: the RAVE algorithm, that estimates the values of the actions of the child nodes to choose from, and selects the one that leads to a better outcome; and the ϵ-greedy algorithm, where instead of always selecting the best node, there's a small probability ϵ that the node will be chosen at random.

Expansion After the selection step follows the expansion step, where one child node is added, thus expanding our search tree.

The selection and expansion steps can be referred to in the literature as the "tree policy": the set of rules that are used to construct and navigate the search tree for finding the most urgent node to expand.

Simulation In the following step, a simulation is run from the newly created node. The rules that govern this simulation are called the "default policy". The simplest default policy would be to make uniform random moves. During the simulation, once a terminal state is reached, a reward is assigned to this state. This step can be referred to in the literature as simulation, rollout, or playout.

Propagation Finally, in the last step the node statistics are propagated back to the parent nodes in the propagation step: the number of visits to each node is updated, and the average reward is recalculated. These statistics will play a role in the node selection and therefore in the decisions made by the tree policy.

Once the Tree Search reaches an end, the optimal sequence of actions can be selected using different statistics: selecting the actions with the highest visit count, choosing the ones with the highest reward, or the ones that maximises UCB, or even a mixture of them.

To conclude, since MCTS uses heuristics to navigate the space of possible solutions, it is a great approach to face high-branching problems, where brute-forcing an optimal solution is not realistic in terms of resources. However, MCTS will not ensure an optimal solution, but a close-to-optimal one in a reasonable amount of time.

1.2 Evolutionary Algorithms

Bio-inspired computing is a field of study that uses methods that imitate biological systems in order to solve computer science problems. One of the main areas of study in bio-inspired computing is Evolutionary Computation (EC), which gets its biological inspiration from the theory of evolution.

One subset of EC is Evolutionary Algorithms (EA), they are used as optimization algorithms, and they work by applying the principles of evolution on a population of potential solutions to a problem at hand. Each of these solutions are defined by a phenotype, and they represent one individual in the population. In biology the term phenotype refers to the set of observable traits of an individual; most of these traits are encoded into the individual's genotype. In a computational setting, the phenotypes are the possible solutions to our problem, and the genotype is the corresponding encoding for the computer to understand.

An EA gets executed in a series of steps [9] as depicted in figure (2).

Initialisation We first need to initialize our population, which can be done through a random initialization. The following steps are consecutive and iterating steps.

Evaluation First, our individuals are evaluated based on a fitness function that rates how they respond to the problem at hand.

Selection Then, based on this assessment, a selection step is performed. Different rule systems exist that can define how to make this choice. The individuals selected become the parents for the next generation.

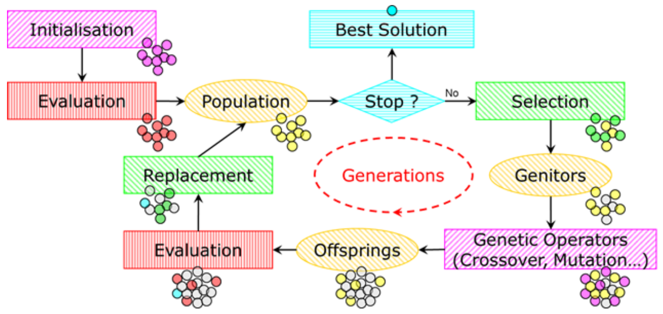

Fig. 2. Steps of an EA

Genetic Operators The third step consists on creating new individuals based on the parents. The most common way to do this is to cross individuals in pairs, and apply the genetic operators of mutation and crossover (or recombination). In biology, two individuals combine their genetic material through sexual reproduction, and the processes of mutation and crossover are the base mechanisms of evolution. A mutation is a change in an individual's genome, and recombination is the process of exchanging genetic material, which is part of the reproductive process. When applied in an algorithmic context, a mutation is a unary operator, it generates one single new genotype when applied to an original genotype. And recombination is a binary operator since it uses two parent genotypes to combine and create one or more offspring genotypes.

There are different types of EA, the most common one is Genetic Algorithms (GA), where the individuals in a population are represented by a sequence of numbers, usually binary. Another type of is Genetic Programming (GP), where every individual is represented by a program or expression; they can be arithmetic operations, mathematical functions, boolean operations, etc. and the data structure they are usually encoded in is a tree. Differential Evolution (DE) is a third type of EA, it is an optimization algorithm for continuous search spaces, in which the individuals to evolve are points in the search space.

To conclude, EAs are used for providing approximate solutions to search and optimization problems that cannot be solved with exact-solution approaches due to the computationally-intensive nature of the problem at hand. Evolutionary techniques are the answer to this type of problem, however, given their random nature, they are never guaranteed to find an optimal solution, but they will find a near-optimal one with the proper amount of time.

2 Integrating EA into MCTS

2.1 EA for the selection step in MCTS

One of the instances where we can find an EA implemented into MCTS is in the selection step during the tree polity, where the formula for selecting the node to expand is optimized through computational evolution. A couple of studies [3, 6] can be found exploring this implementation.

In the first paper [3] they use GP to evolve the UCT formula for strengthening the MCTS algorithm. As explained before, UCT is used to select which node to expand, addressing the exploration/exploitation dilemma. Evolving this formula can improve the heuristics that go into the exploitation part of this process.

For evolving the UCT expression, our individuals in the population are mathematical expressions in the form of a tree. Expression trees are binary trees where the internal nodes correspond to operators and the leaf nodes to operands.

In order to select the best expressions to reproduce for the next generation they make use of the Swiss tournament as the selection technique in the EA. This consists in confronting individuals two by two, the winner gets a point, and individuals with the same score get to compete against each other. At the end, fitness is assessed taking into account the number of wins and the overall scores of their opponents. This fitness score is used to sort the individuals from better to worse performing: the lower half of the individuals do not reproduce, and the other half get to reproduce more the higher their score is. On top of this, a mutation operator can be applied to every organism with a certain probability so a leaf of an individual is replaced by another random leaf.

When comparing MCTS with the evolved UCT expression against the regular UCT,the evolved one reaches a better solution 86% of the time, and when comparing it to RAVE, is better 70.5% of the time.

The other study [6] also evolves the UCT expression, but using smaller population sizes in order to make faster decisions. In a similar manner to the previous study, they start with the UCT formula as a parent to the first generation, and run the Evolutionary Algorithm to get an evolved UCT. Apart from this EA-MCTS, they implemented a Semantic-Inspired EA-MCTS (SIEA-MCTS) to promote diversity in the expression population.

In the end, SIEA-MCTS and EA-MCTS both were better-performing when compared to standard MCTS (with UCT) and MCTS-RAVE.

2.2 EA for the simulation step in MCTS

Another instance where EAs can be found integrated into MCTS is during the simulation step. A couple studies [1, 8] propose different solutions to approaching this.

The first paper [1] uses GP to evolve an evaluation function that guides MCTS' random moves through the simulation step. The reasoning behind improving the default policy is that uniform random simulations may not be an accurate representation of the value of a leaf node, so changing the default policy to a non-random one may help in a faster convergence of the algorithm.

Through the MC tree search, the evolved evaluation function is used on every move of the simulation to evaluate a number of random moves and select the best ones until a final state is reached.

In a similar fashion to the previous papers, here, they used tree-based GP. For the evolution of the formulas, other genetic operators apart from the usual crossover and mutation where used to improve the GP approach. For instance, selective crossover, where, when crossing two parents, there is a random chance of applying one-way crossover (where only one parent exchanges its genes), or two-way crossover (the classical approach). Also, local mutations are applied with a certain probability: in tree-based individuals, mutations can have a great effect on their performance, radically altering its fitness. To be able to still apply the mutation operator without having too much of an impact, mutations are applied locally only to leaf nodes (operands of the expression) with a numerical value. And the mutations consist on multiplying the value of the node with a factor. Traditional mutation is, however, still applied. And the choice between traditional or local mutation is taken with a defined probability.

They assessed the fitness of the individual with a coevolution round, where members of the population are pitted two by two and assigned 1 point for a win and 0.5 for a draw. To select the individuals that will procreate in the following generation, random subsets are taken from the population, and out of those subsets, the fittest individuals are selected.

To evaluate their algorithm, a benchmark was performed using Evo-MCTS for playing board games and comparing against a counterpart algorithm (standard MCTS, or Alpha-Beta for low branching games). 3 different board games were tested with different levels of complexity and number of playouts, and in all of them, Evo-MCTS showed a better performance than its counterpart.

In the second study [8] they used a different approach where their population of individuals to evolve consists on a vector that controls and biases the moves made during the simulation step.

To get this vector, the MCTS is run, and on each iteration the vector is used in the default policy, the performance of each vector is evaluated, and at the end, the best-performing one is selected as the control vector.

To further explain this, a GA is coupled with MCTS and on every rollout all the individuals from our population of vectors get their fitness value evaluated. This study is performed in a game-playing context, so the fitness of an individual comes from the scoring statistics of the game MCTS is run to play, and so, these statistics rate how well the MCTS algorithm performs. In this study in particular, they just take into account the mean score. The biased default policy is sampled a number of times to get an accurate representation of the mean.

To assess the improvement in performance of this technique, MCTS was run with: uniform random rollouts (the standard MCTS), evolving biased rollouts (the method described above), and pre-evolved biased rollouts (where the one control vector is fixed during the tree search, but it is the product of an evolution process as described before). The mean score of the MCTS with evolved rollouts shows a slight improvement over the random uniform rollouts, and the pre-evolved one shows a significant performance boost.

Summing up, the use of EA to improve the selection step in MCTS, and to guide MCTS rollouts, seem to be two valid approaches for improving the classical implementation of MCTS. In all four cases, a comparison with the regular MCTS showed an increase in performance.

3 Integrating MCTS into EA

The opposite implementation where MCTS is used to improve Evolutionary Algorithms is not yet a technique thoroughly explored in the scientific literature. However, we can see it being implemented in two recent papers [5, 10].

We can find a couple instances of this implementation in the former paper [5]. More specifically this study delves onto the Rolling Horizon (RH) algorithm, an optimization technique used when short-term decisions need to be taken while there is a long-term strategy. This strategy makes sense in the context of video-games, where small quick decisions need to be taken with the final goal of winning the game. In this paper, an EA is used to evolve a vector that represents the sequence of states in the playthrough of a game. In the case of an RH EA, the vector of states needs to be evolved at each tick of the game (the period of time consumed to make an action).

On every EA generation, the vector is evolved and evaluated, and then the fittest one (the one leading to the best game outcome), is selected to define the subsequent action from the current state S_t to the following one S_{t+1}.

3.1 MCTS for the initialization step in EA

The first implementation of MCTS into EAs is in the EA's initialization step: they run a classic MTCS with UCB for 1/2 a tick of the game, and the sequences of actions in the MCT that lead to the best outcomes are selected to be the genotypes for the first generation.

3.2 MCTS for the evaluation step in EA

The second instance where MCTS is combined into EA is for the fitness evaluation step. For evaluating the EA vectors, MCTS rollouts are simulated to add more robustness to the calculation. Without the MCTS rollouts, the fitness of the vector is calculated after evaluating every state of the vector with a heuristic function, and then the final value of fitness can be calculated in many ways using these evaluation values (ev): $ev_{final} - ev_0$, $max(ev)$, $min(ev)$, $sum(ev)$, etc. or even a mixture of them. When adding the MCTS rollouts we just add more states and their corresponding evs to the calculation of the fitness, with the benefit of having a more confident estimation.

3.3 MCTS for the selection step in EA

In the other research paper that implements MCTS into EAs [10] their aim is to improve a DE algorithm for solving multimodal optimization problems. The role that MCTS plays in this process is for selecting the most promising regions in the search space of the DE algorithm, so that it can later evolve points from these regions to find solutions to the optimization problem.

There is an iterative loop between the MCT and the DE algorithm. For the first iteration, an MCT is constructed where the root node represents the entire solution space of the problem of interest. The tree grows partitioning a parent node into two child nodes using k-means clustering, making this MCT a binary search tree. After this, the nodes are selected using UCB with an ϵ-greedy approach. And so, the selected node corresponds to the subspace of candidate solutions with the most promising regions. Subsequently, the DE algorithm evolves the new candidate solutions generating new ones that are then backpropagated through the MCT, updating the values of the nodes.

In short, this MCT-EA is similar to a classical MCT where the simulation step is the search process by the DE algorithm, that guides the growth of the tree. On the flip side the DE algorithm evolves its population of solutions efficiently thanks to the selection of promising regions of the search space by the MCT.

To assess the performance of MCT-EA, they used a multimodal optimization test suite. Their results showed that their new implementation outperformed other algorithms on 12/13 multimodal functions to optimize.

Apart from these two papers implementing MCTS into EA, not much research has been done on this topic. However, some other studies can be found on Reinforcement Learning-based EA that could inspire other implementations of, specifically, MCTS into EAs. For instance, one study [4] where they use Q-learning, a reinforcement learning technique, for optimizing the Crossover and Mutation operators of a Genetic Algorithm.

To sum up, when it comes to ameliorating EA optimization algorithms with MCTS, even though the research has not been extensive in this area, we can find three different instances where it has been implemented: as a population initialization technique, for increasing robustness in fitness calculation, and to filter the search space in DE. All of them exhibited some kind of improvement when compared to a classical EA. In a broader aspect, approaches using other RL techniques can be found in the literature, and this can be a starting point for inspiring new MCTS-improved EAs.

4 Conclusion

Different approaches can be taken when combining MCTS and EA, whether it is using MCTS to improve the optimization process in an EA, or using EA for enhancing the search algorithm in MCTS. One can think of an optimization process as a search algorithm because they follow essentially the same paradigms: an optimizer just needs to search the optima in a solution space. The approaches presented in this work could be used to tackle similar problems, and they all showed that combining the heuristic nature of MCTS and EAs, with the precision of a search tree in MC, and the robustness of EAs, can lead to new and better-performing optimization techniques.

References

1. Benbassat, A., Sipper, M.: EvoMCTS: A Scalable Approach for General Game Learning **6**, 382–394 (2014). https://doi.org/10.1109/TCIAIG.2014.2306914
2. Browne, C.B., Powley, E., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., Colton, S.: A Survey of Monte Carlo Tree Search Methods. IEEE Transactions on Computational Intelligence and AI in Games **4**, 1–43 (2012). https://doi.org/10.1109/TCIAIG.2012.2186810
3. Cazenave, T.: Evolving Monte Carlo tree search algorithms. Dept. Inf., Univ. Paris **8** (2007), publisher: Citeseer
4. Chen, Q., Huang, M., Xu, Q., Wang, H., Wang, J.: Reinforcement Learning-Based Genetic Algorithm in Optimizing Multidimensional Data Discretization Scheme. Mathematical Problems in Engineering (2020)
5. Gaina, R.D., Devlin, S., Lucas, S.M., Perez-Liebana, D.: Rolling Horizon Evolutionary Algorithms for General Video Game Playing (2020)
6. Galván, E., Simpson, G., Ameneyro, F.V.: Evolving the MCTS Upper Confidence Bounds for Trees Using a Semantic-inspired Evolutionary Algorithm in the Game of Carcassonne (2022). https://doi.org/10.48550/arXiv.2208.13589, arXiv:2208.13589 [cs]
7. Grefenstette, J.J., Moriarty, D.E., Schultz, A.C.: Evolutionary Algorithms for Reinforcement Learning. Journal of Artificial Intelligence Research **11**, 241–276 (1999). https://doi.org/10.1613/jair.613
8. Lucas, S.M., Samothrakis, S., Pérez, D.: Fast Evolutionary Adaptation for Monte Carlo Tree Search. In: Applications of Evolutionary Computation. pp. 349–360. Springer (2014). https://doi.org/10.1007/978-3-662-45523-4$_2$9
9. Vikhar, P.A.: Evolutionary algorithms: A critical review and its future prospects. In: 2016 International Conference on Global Trends in Signal Processing, Information Computing and Communication (ICGTSPICC). pp. 261–265 (2016). https://doi.org/10.1109/ICGTSPICC.2016.7955308
10. Xia, H., Li, C., Zeng, S., Tan, Q., Wang, J., Yang, S.: Learning to Search Promising Regions by a Monte-Carlo Tree Model. In: 2022 IEEE Congress on Evolutionary Computation (CEC). pp. 01–08 (2022). https://doi.org/10.1109/CEC55065.2022.9870281