Artificial Intelligence 4 Games

Genetic Algorithms for Single-Player Game Tree Search

2020

Jakub Kowalski, University of Wrocław

Open Loop vs Closed Loop Tree Search

Closed Loop

In deterministic scenarios, a game tree can store future game states in its nodes.

Thus, when traversing a sequence of moves, it is enough to retrieve a state without the need to apply them once again. When adding a new node to the tree, we may simply apply a proper action from the parent node calling a forward model once.

Works well when simulating game is expensive and we have memory to store the states.

Open Loop

The idea is that the nodes do not store the states, but only the statistics.

Forces the use of forward model every time an action is chosen within a tree.

Works well if simulator is fast, and makes states stored within a tree quickly outdated. Or in a case of stochastic games, where states drawn during each simulation reflects the probabilities from the game, and allows to averaging when a sufficient number of samples is gathered.

Genetic Algorithms for Game Tree Search

Relatively novel, but actually a very straightforward idea of using Genetic Algorithm as a game tree search algorithm.

Evolution is used in the same manner as MCTS uses rollouts and game simulator. An agent evolves a plan, performs the first action of this plan, and then evolves a new plan repeatedly. Evolution runs online, and it has to return a plan within a limited, usually turn-constrained, time.

“Rolling Horizon” name comes from the fact that planning happens until a limited lookahead in the game, so we have to keep on replanning at every time step.

This generic description, can be employed in various algorithm versions. Thus, we treat *Rolling Horizon Evolutionary Algorithm (RHEA)* / *Online Evolution* / *Online Evolutionary Planning (OEP)* / *other fancy name* rather as a family of algorithms than a specific implementation.

In this lecture we will present example choices that can be used for a specific parts of the algorithm.

Because evolution is naturally more suited to handle single-player games, so we start with this type of games. To handle multi-player ones, some opponent prediction is additionally required, in a form of e.g. coevolution.

Rolling Horizon Evolutionary Algorithm

Literature

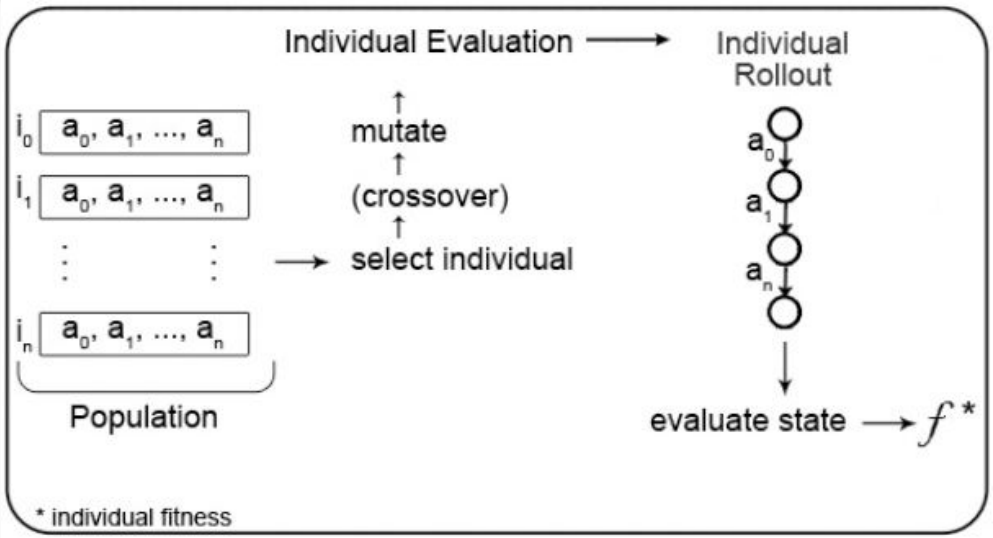
- Perez, D., Samothrakis, S., Lucas, S., Rohlfshagen, P: Rolling horizon evolution versus tree search for navigation in single-player real-time games. GECCO, pp. 351-358, 2013.
- Samothrakis, S., Roberts, S. A., Perez, D., Lucas, S. M.: Rolling horizon methods for games with continuous states and actions. CIG, pp. 1-8, 2014.
- Perez Liebana, D., Dieskau, J., Hunermund, M., Mostaghim, S., Lucas, S.: Open loop search for general video game playing. GECCO, pp. 337-344, 2015.
- Gaina, R. D., Perez-Liebana, D., Lucas, S. M., Sironi, C. F., Winands, M. H. Self-Adaptive Rolling Horizon Evolutionary Algorithms for General Video Game Playing. COG, pp. 367-374, 2020.

RHEA

Our individual (genotype) is a sequence of moves. As a phenotype we usually consider the game state after application of such sequence.

In contrast to MCTS, RHEA works on a limited horizon, thus requires a heuristic evaluation function. (Note however, that this function may be implemented as a batch of random simulations...)

Apart from a few specific problems that might (but does not have to) occur, in most cases we can just behave as during a standard evolution.



The important aspect is that the evolution needs to work in a very restricted time limit.

Rolling Horizon

After the best action sequence for a current turn has been found, its first action is performed. Then, there are multiple ways how we can proceed to the next game turn.

We can discard the entire previous-turn population and start new evolution from the newly initialized individuals.

We can keep the entire population, but “roll” it. Thus, remove the first gene (action) from each individual, and add a new one at the end - e.g. straightforwardly randomizing a next legal action.

The latter option is probably more widely used, as it keeps the already gathered knowledge. Additionally, my may also discount the values of all rolled individuals.

Of course, there are many middle-ground options, that allow us to reuse some knowledge, but use turn end as a chance for increasing exploration and prevent the algorithm to search too narrowly.

Thus, we can keep and roll only a part of the population, and substitute the rest with newly initialized individuals.

Initialization

Although initialization from the random sequences of action is computationally cheap and usually works well, it may be worth to consider also other options.

These ones, more expensive, provide a better start that may significantly decrease the number of generations needed to obtain a high-quality solution.

When it is worth to try them? If random initialization struggle for a long time to obtain a solution of a quality achieved by the other methods. If comparing computational effort shows it would be wiser to switch - do this.

1SLA

One Step Look Ahead strategy is to first greedily create a locally-optimal individual by exhaustive search of all possible actions at each gene and choosing the best one according to the heuristic. The remaining population is created as a mutated variants of this one.

MCTS

Run MCTS for a fraction of first turn time, and create the first individual by greedily traversing the created tree. The rest individuals are mutated from the first.

Evaluation

Assuming all game states are evaluated using a heuristic evaluation function, there are number of ways they can be used to evaluate a sequence of actions

- ➔ Keeping the value of the last game state reached
- ➔ Keeping the difference between the values of the last and the first game state (improvement)
- ➔ Keeping the average of all game state values
- ➔ Keeping a discounted sum of all values
- ➔ Keeping maximum/minimum value of visited states

We may also adapt MC-based approach and use simulations in addition / instead of the heuristic function.

A common way is to run a batch of rollouts deepening the search by some predefined number of actions and calling the evaluation function when they end.

Running full rollouts until the game's end is in most cases too weak evaluation given the limited time for gathering the statistics per one individual.

Mutation

As in standard EA, there are many available mutation operators including, e.g.

- *uniform mutation*, changing each gene with a given probability (e.g. 1/L where L is the length of the individual)
- *n-bit mutation*, changing n random genes
- *diversity mutation*, we can remember values explored for all genes and mutate the least explored gene modifying it to its least visited value.
- *softmax mutation*, biasing mutation towards the beginning of the genome, where changes most affect the phenotype

$$\text{Softmax}(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

Legality issues

There are two potential problems: the value for a mutated gene should be a legal action, and the following actions after the change should also remain legal.

One solution, costly, is to modify the operators to include only legal actions (requires knowing them!), and check the rest of the genome fixing (i.e. re-randomizing) all the actions that requires that.

Second approach, risky, is to not care, and somehow evaluate such broken individuals, by e.g. evaluating their last legal game state or giving them low value.

Crossover

Standard uniform crossover or n-point crossover can be used here as well.

The problematic case is when the offspring is an illegal sequence of actions.

There are few ways to deal with this problem, and they heavily depend on the probability of such situation.

Note that the operator-choice matters here. 1-point crossover is usually the safest, and also the most reasonable given the action-sequence semantic. Merging good ‘beginnings’ with good ‘endings’ makes sense in many types of games.

We can just skip illegal children, dynamically reducing the size of the offspring population, if it is rare.

We can also continue breeding procedure until full-sized offspring population is computed.

Last but not least, we may fix illegal individuals, similarly as in case of mutation operators.

Search Horizon

In RHEA, the search horizon is clearly stated as it equals to the genotype length.

In standard evolution, when estimating computational resources the problems have fixed length and we consider population size versus number of generations tradeoff.

Here, we have additional parameter that influences the computing cost and the quality of obtained solutions.

Its of course very game-dependent and requires careful testing and observing the in-game behaviour of an agent. Also it is correlated to the type of evaluation used.

Usually, we need to consider computational cost, behaviour of the enemies, and the game's structure. E.g. what is the smallest horizon that allow us to evade an obstacle when we see it in a racing game?

Also, not always the longer horizon the better.

The algorithm for dynamic adjusting the length can be found in the following paper. The idea is based on observing the flatness of the reward landscape.

★ R. D. Gaina, S. M. Lucas, D. Perez-Liebana. Tackling Sparse Rewards in Real-Time Games with Statistical Forward Planning Methods. AAAI, vol. 33, pp. 1691–1698, 2019.

Some additional enhancements

Literature

- ★ The same :-)

Continuous States and Actions

Tree-based approaches usually struggle with continuous time and space, as they have to be somewhat discretized to allow finite tree representation.

Of course, everything that works for MCTS works also for genetic algorithms, but in GA-based approach we have even less restrictions, as we do not require to build the tree directly, and are not so dependent on revisiting the same paths.

Note that potential infinity of e.g. action space is not a problem for GAs. Assuming we can draw next legal action with some probability, all the standard variation operators remain compatible.

Mutation usually just substitutes an action with a new one (or adds an offset), while crossover copies action across different individuals.

Depending on a domain, some new better-suited operators may be introduced, e.g. halving a value of some gene, or using some weighted sum (e.g. based on a random weight) towards parent values.

Combining RHEA with Tree Search

Fitness values can be calculated by executing the sequence of actions until all actions are executed or a terminal state is reached.

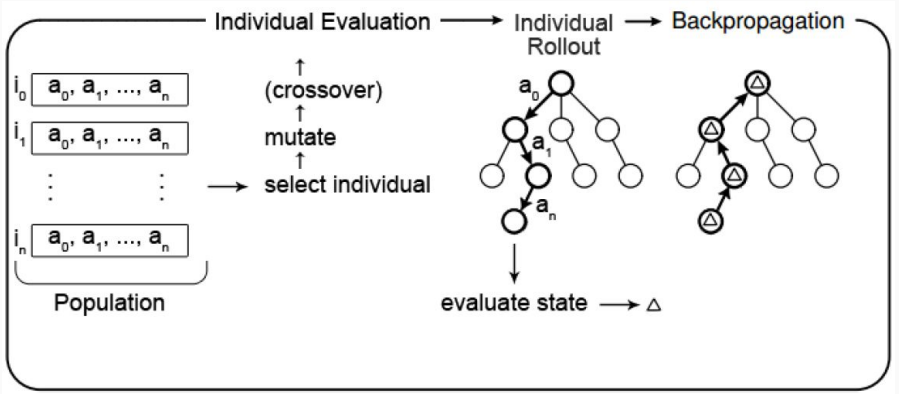
While evaluating an individual, a tree is generated with the actions performed.

When some of these sequences are repeated in different individuals, the tree nodes will be reused to gather the stored statistics.

This provides the algorithms with a way of calculating statistics about the actions taken, as well as reusing information from one game step to the next by keeping the game tree.

This works with the fitness being e.g. the average value of all nodes visited during the evaluation of the individual.

This approach takes take advantage of the statistics stored in the game tree.



Anytime approach

Usually RHEA is approximating anytime algorithm, with batched-approach, requiring to compute a complete generation to stop.

For this reason, it is often handy to reduce the size of the population, allowing faster population updates.

We may go extreme and use a hill-climber like (1+1) ES, but this makes us lose benefits of population-based approach.

Alternatively, to make RHEA truly anytime, we may use slightly different approach. Instead of reducing the population size itself, we may restrict the updates and create only one new individual per generation.

For example, at each generation we randomize whether to use crossover or mutation. Then we apply standard selection + breeding/mutation operators and evaluate the new individual. If its fitness is better than the worst one in the population - we replace it.

Choosing the right operators and parameters

How

1) Experience
2) Testing
3) Analysis
4) Testing
5) Gut feeling
6) Testing
7) Experimentation
8) Testing
9) Testing
10) Testing

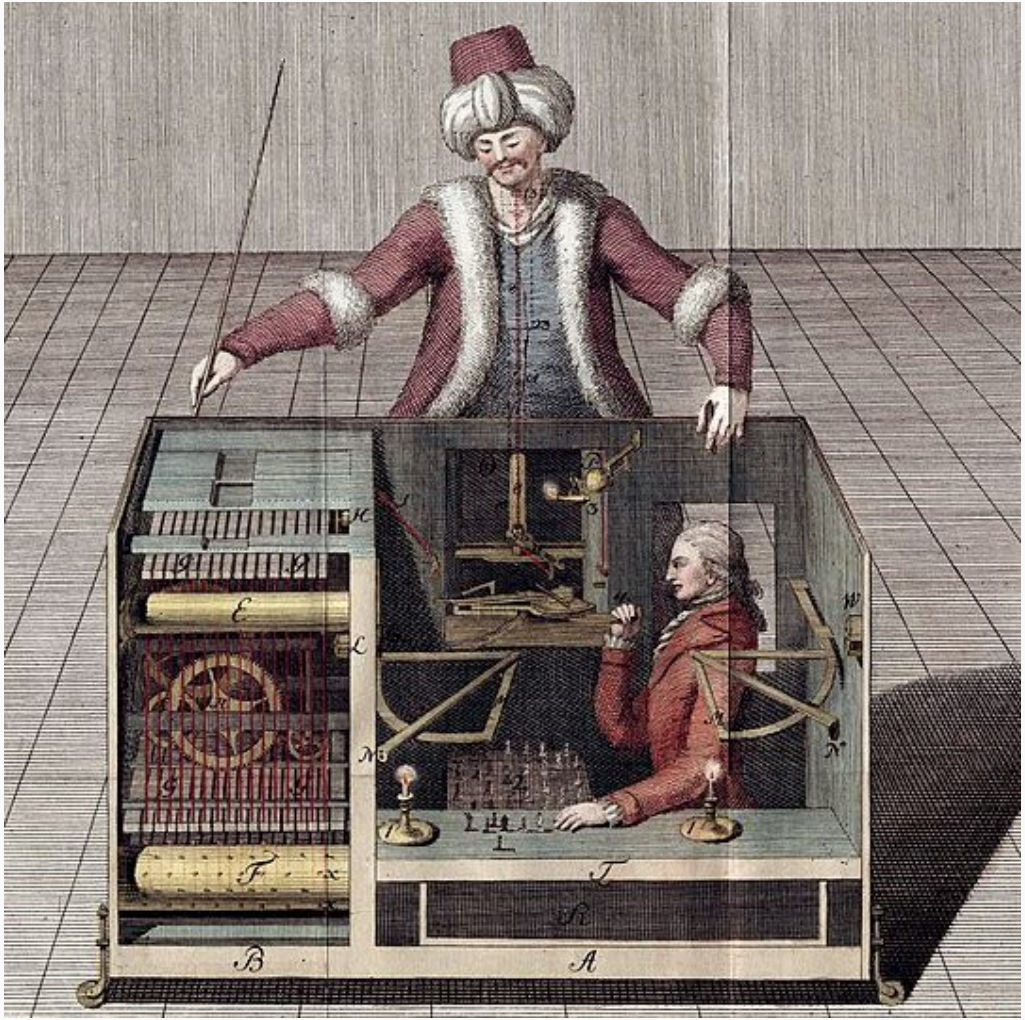
Assuming we do not have any deeper knowledge about a problem we are about to tackle, it is worth to start with a standard setting.

➔ Random initialization, one-point/uniform crossover, uniform mutation, any fitness-based selection.

Then, by analyzing the behavior of the algorithm - both on a charts (how the populations progresses) and in a gameplay (actions of the character) we may deduce what changes are needed to improve performance.

It is probably easier to first set approximate values of important variables such as population size, chromosome length, and try to tune operators and other parameters later.

Thanks!



Bonus reference quiz