Hello! My name is Raluca Gaina and I am a PhD student at Queen Mary University of London. And in this video I'll be talking about a self-adaptive rolling horizon evolutionary algorithm for general video game playing – talk based on a paper published at the IEEE Conference on Games 2020. To quickly unpack the title:

- rolling horizon evolutionary algorithms are algorithms which use concepts from evolution to evolve action plans while playing games;
- self-adaptive refers to the algorithm modifying its parameters while playing games in order to improve its structure as well as the action sequences
- and general video game playing is the domain concerned with creating artificial players able to play any game possible.

We're looking at this topic in particular as there have been many advances in recent years in evolutionary algorithms for game-playing and an ever increasing number of parameters controlling its thinking process. But since we're trying to play a variety of games, some of these parameters work well only on some games, or even in particular situations. So we are trying to increase the adaptability of the player by allowing it to modify its control parameters while running.

Authors

Raluca D. Gaina

Diego Pérez Liébana

Simon M. Lucas

Queen Mary University of London
United Kingdom

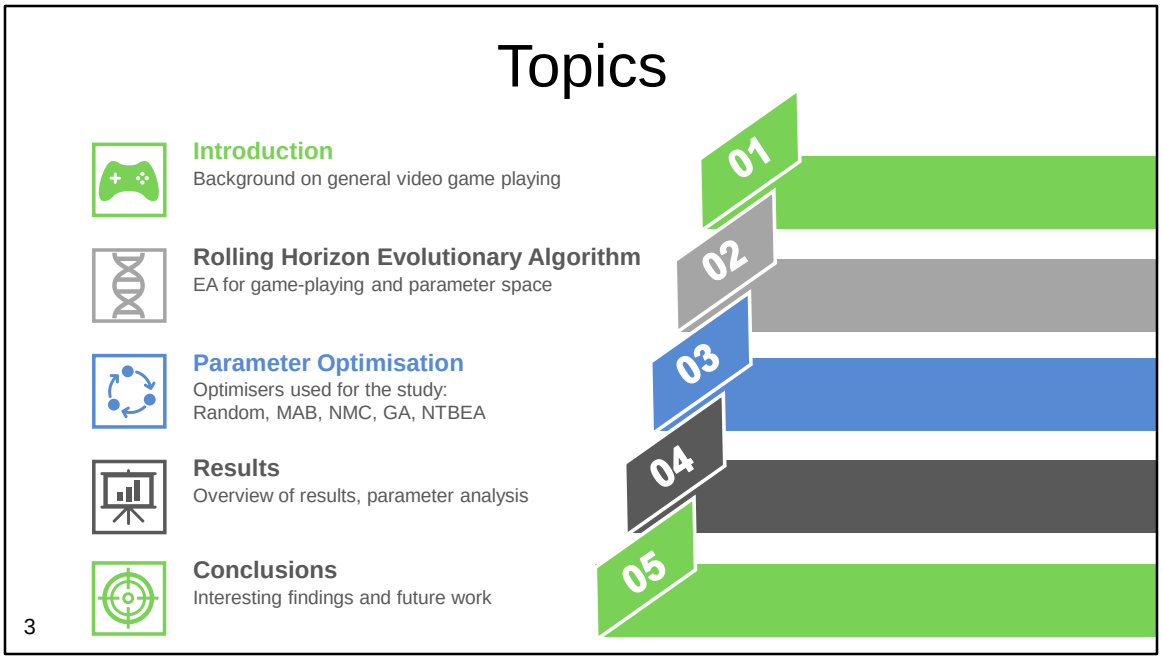
Chiara H. Sironi

Mark H.M. Winands

Maastricht University
The Netherlands

2

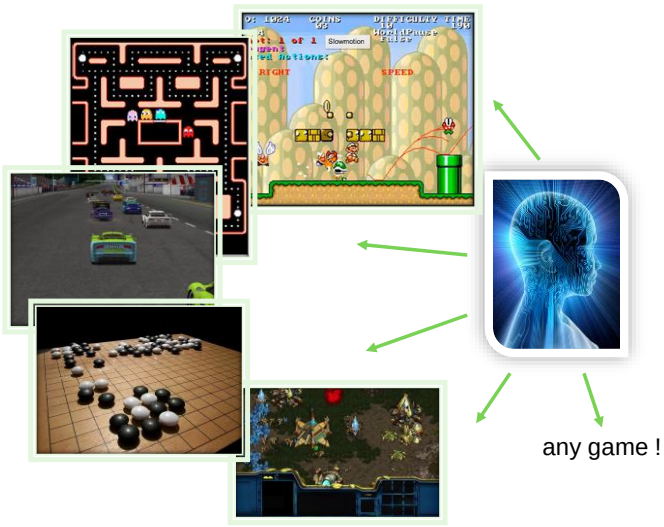
This work was done in a collaboration with colleagues from Maastricht University.



To give a quick overview of this video, we're looking first at some background on general video game playing to contextualize the work. Next, I'll go over how our rolling horizon evolutionary algorithm works and its parameter space that we are concerned with in this work, as well as the parameter optimisers used for this study. Lastly, we'll look at the results obtained, some interesting insights and a few ways in which this work can be extended further. Let's dive in!

Introduction

General Video Game Playing



Adaptive players that can act intelligently in any given game, even those unseen before or with unknown rules

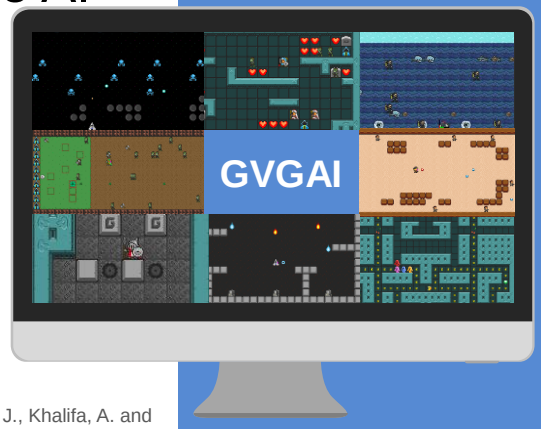
5

This work is within the domain of general video game playing. We've seen a lot in media news of very good AI players in Go, Dota, Starcraft. And while these are very impressive advancements, those players would have a very hard time picking up a new game. Therefore we're looking instead at highly adaptive general players, that are able to take any game given to them and act intelligently in that environment.

Benchmark:
General Video Game AI

www.gvgai.net

- **Single-player planning**
- Two-player planning
- Single-player learning
- Level generation
- Rule generation



Pérez Liébana, D., Lucas, S.M., Gaina, R.D., Togelius, J., Khalifa, A. and Liu, J., 2019. **General Video Game Artificial Intelligence**. Synthesis Lectures on Games and Computational Intelligence, 3(2), pp.1-191.

6

We're using General Video Game AI (or GVGAI) for our benchmark, which contains over 160 games and proposes many different challenges for AI methods, from playing games by planning without seeing the game previously, learning how to play games by repeatedly playing them and gathering experience; or even playing two-player games, generating levels or generating rules! In this particular work, we're focused on the single-player planning track. You can find out about the framework and competition in the book mentioned at the bottom of the screen.

Game Set

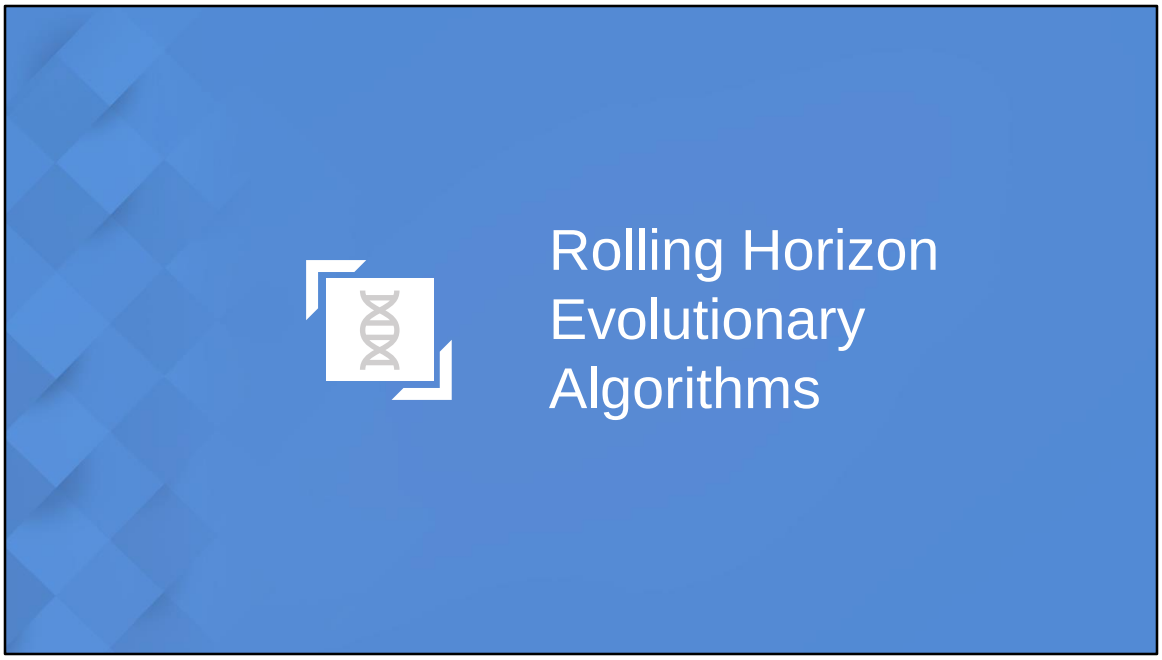
- 5 levels each
- Different winning conditions
- Different sprite interactions
- Different scoring systems
- Different levels of stochasticity

Dig Dug
Lemmings
Roguelike
Chopper
Crossfire
Chase
Camel Race
Escape
Hungry Birds
Bait
Wait for Breakfast
Survive Zombies
Modality
Missile Command
Plaque Attack
Seaquest
Infection
Aliens
Butterflies
Intersection

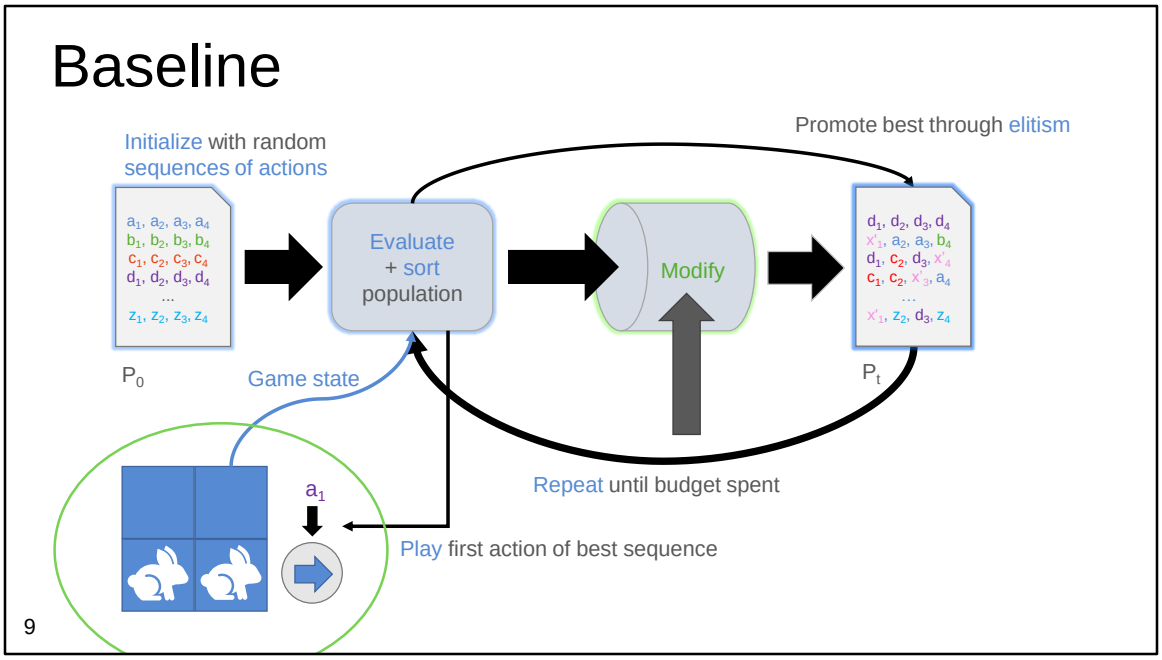
Raluca D. Gaina, Sam Devlin, Diego Perez-Liebana, Simon M. Lucas, 2020. **Rolling Horizon Evolutionary Algorithms for General Video Game Playing**. In arxiv:2003.12331.

7

We use a subset of 20 games from the GVGAI corpus with various features. All are listed on the right, with some highlighted that will be mentioned more in this discussion. A breakdown of all their features can be found in the paper cited at the bottom, and all games can be found and played from the GVGAI framework. For our experiments, we used all 5 levels of each game, with 20 runs per level (that gives 100 runs per game) and results and statistics will be presented averaged over all runs in a game.



Within this domain, we look at rolling horizon evolutionary algorithms, which have been shown to achieve high performance on a range of games, beating the previous state of the art.



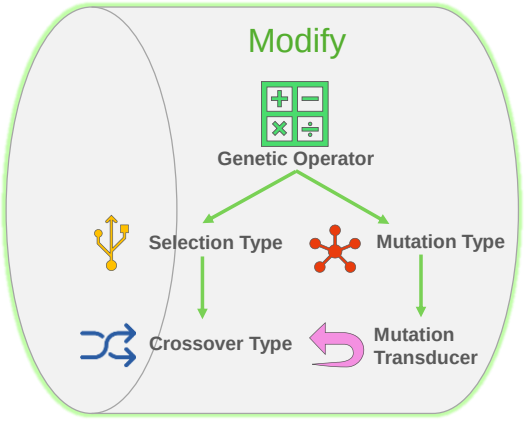
The algorithm runs at every game tick, and begins with a population of individuals, where each individual (or row in this diagram) is a random sequence of actions to be played in the game. We can call the initial population P-0. We then evaluate each individual using a copy of the current game state observation and a forward model (which allows to simulate the effect of actions without actually playing them); the game state reached after running through all the actions in the sequence is evaluated with a heuristic function and the given value becomes the fitness of the individual. In this case, the heuristic function simply aims to maximize the game score obtained by the agent, prioritizing winning game states and avoiding losing game states.

Based on these values, the population is sorted and the best individual is promoted directly to the next generation through elitism. The population is then modified with various genetic operators in order to obtain a new population of the same size. The process is repeated until the budget is spent, in this case 1000 simulations with the forward model. Finally, the first action of the best sequence at the end of this evolution process is actually played in the game. Everything repeats in the next game tick.

Many parameters control this process, but for this work we are interested in those

that most affect one iteration, thus those included in this part. Let’s look at them more closely.

Parameters (1)



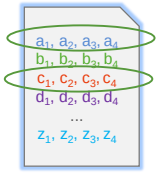
Genetic Operator

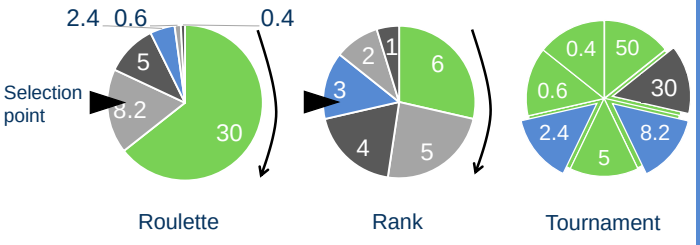
- Crossover only
- Mutation only
- Both

10

We have 5 parameters that we will be automatically adjusting during play in this work, with some dependencies. The Genetic Operator parameter controls which operators are applied to generate the offspring: if crossover is used, the left branch only is explored and new individuals are not mutated; if mutation is used, the right branch is explored instead and each individual in the population is mutated once to create the next generation. Alternatively, both can be used, in which case individuals are created through crossover and then mutated as well.

Parameters (2)





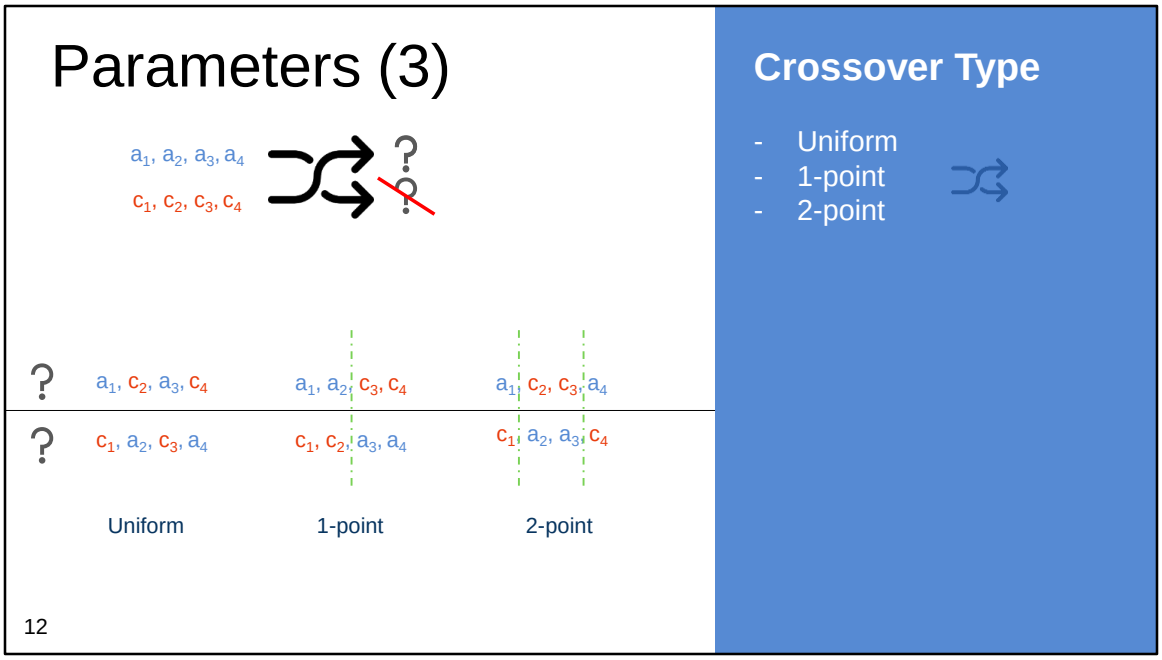
Selection Type

- Roulette
- Rank
- Tournament

11

Selection is the process of selecting the parents in case crossover is used. Three different types are available: roulette, which selects individuals with probabilities directly proportional to their fitness; rank, which first ranks individuals based on their fitness in descending order and selects them with probabilities directly proportional to their rank (aiming to minimize the impact of large differences in fitness). And tournament, which randomly chooses a subset of the population and then the best amongst these is selected.

The selection type chosen is repeated so that two individuals are selected.



Next, the two individuals, in this case the red and blue ones, are crossed to obtain offspring. Three options are available here as well: uniform crossover randomly selects genes from the parents with equal probabilities. 1-point crossover randomly finds an index in the individual and chooses the first part (up until that index) from one parent, and the second part from the other. And 2-point crossover find 2 indexes and alternatively chooses the sections from the parents.

In our implementation, one individual is randomly discarded, so only one new individual is created from a crossover operation.

Parameters (4)

a_1, a_2, a_3, a_4 → ?

(Random new gene)

? a_1, a'_2, a_3, a'_4 **Uniform**

a'_1, a_2, a_3, a_4 **1-bit**

a'_1, a_2, a'_3, a'_4 **3-bit**

- **Softmax**: change 1 gene towards beginning of individual
- **Diversity**: change least visited gene to its least visited value

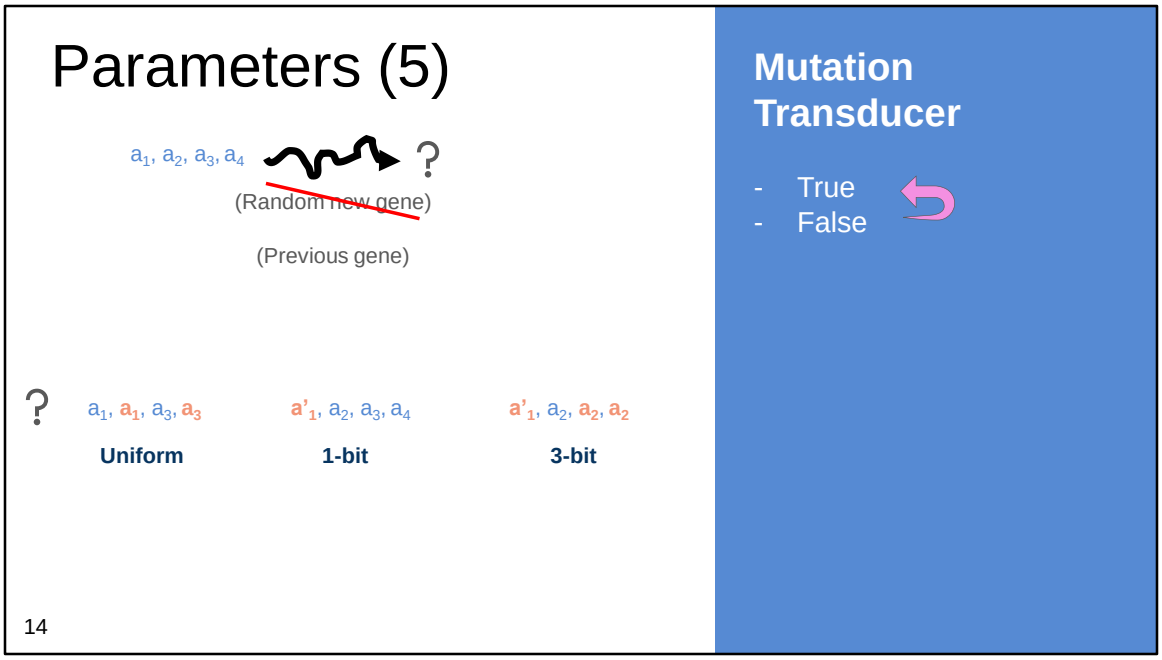
13

Mutation Type

- Uniform
- 1-bit
- 3-bit
- Softmax
- Diversity

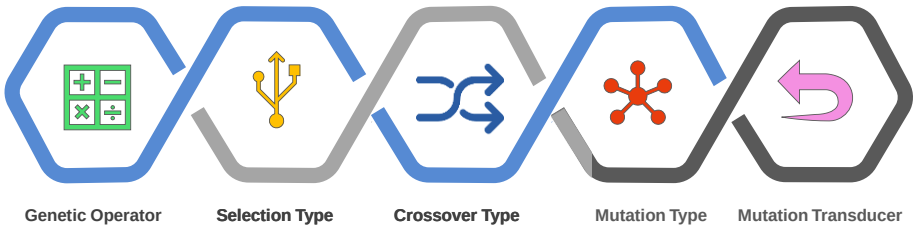
Mutation is the process of modifying some genes in an individual to new random values. We have 5 options here to select which genes are modified:

- Uniform mutation modifies all genes, each with probability 1/L, where L is the length of the sequence
- 1-bit mutation modifies one randomly chosen gene
- 3-bit mutation modifies 3 randomly chosen genes
- Softmax mutation uses the softmax equation to bias the gene selection towards the beginning of the individual, which causes the largest perturbance in phenotype (or, the actual behaviour of the agent in the environment) (as the overall path the agent takes through the level would be most different if the beginning of that path changes)
- And lastly, diversity mutation changes the least visited gene to its least visited value, to promote exploration of all genes and their values.



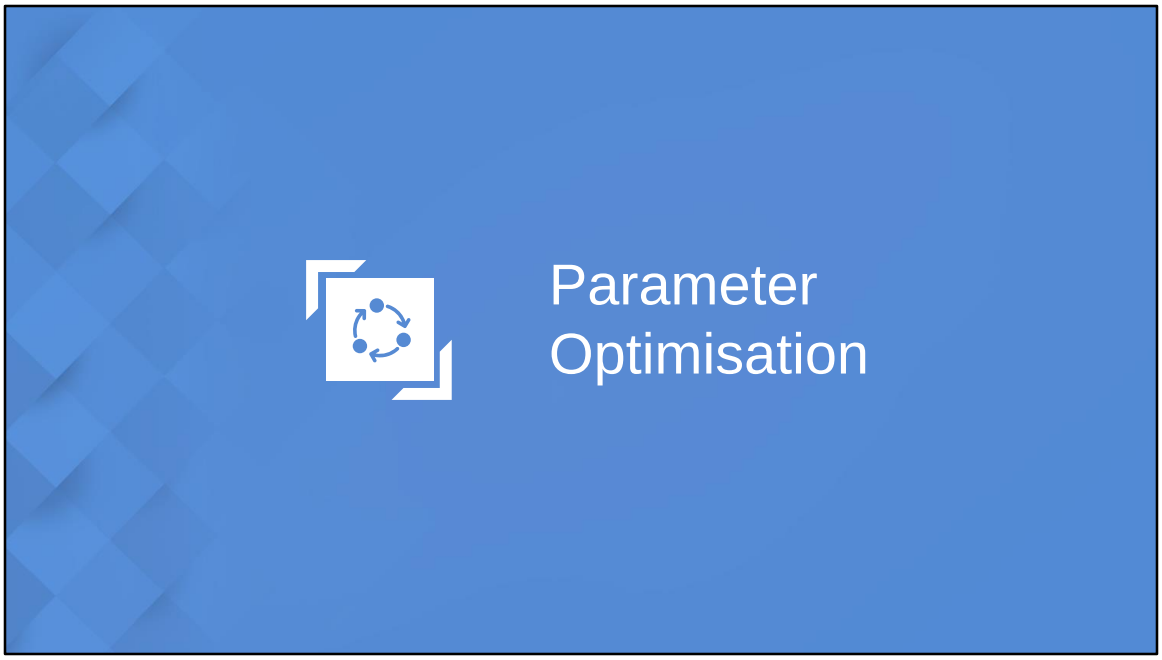
The last parameter is the mutation transducer, which can only be enabled or disabled. If enabled, this parameter causes the genes to be changed to the previous action in the sequence, instead of a new random one. So uniform, 1-bit and 3-bit mutation would produce something like this instead, with actions repeated. This is meant to address the jitteriness observed in many of these general agents, which often choose different directions to go in at every game tick, especially in environments with sparse rewards; now, the agent is more likely to repeat the same action several times and thus choose a direction and stick to it.

Parameters (6)

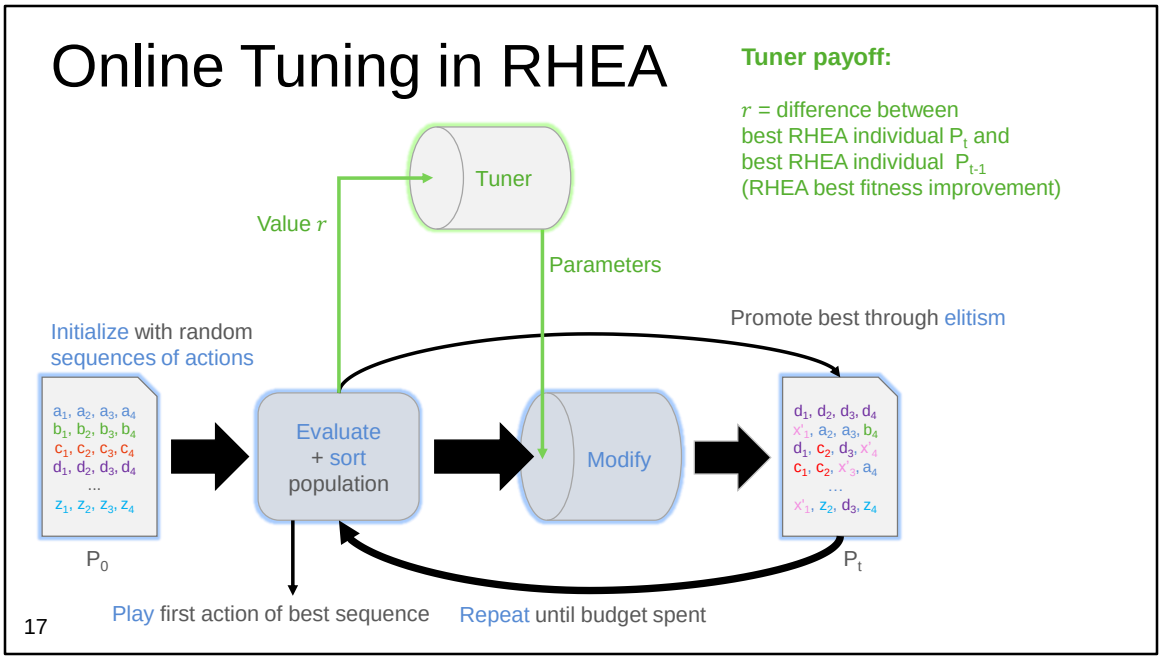


15

As a recap, these are the parameters we're looking at:.

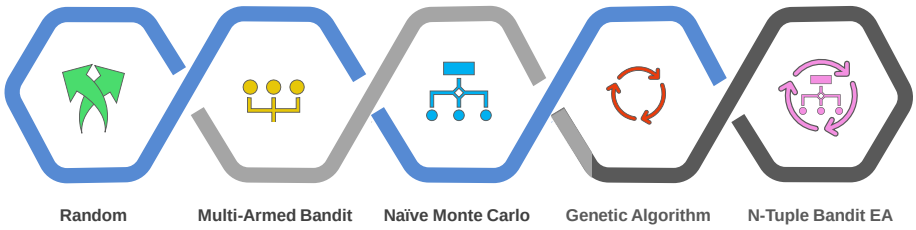


So how does automatically modifying these fit into the algorithm?



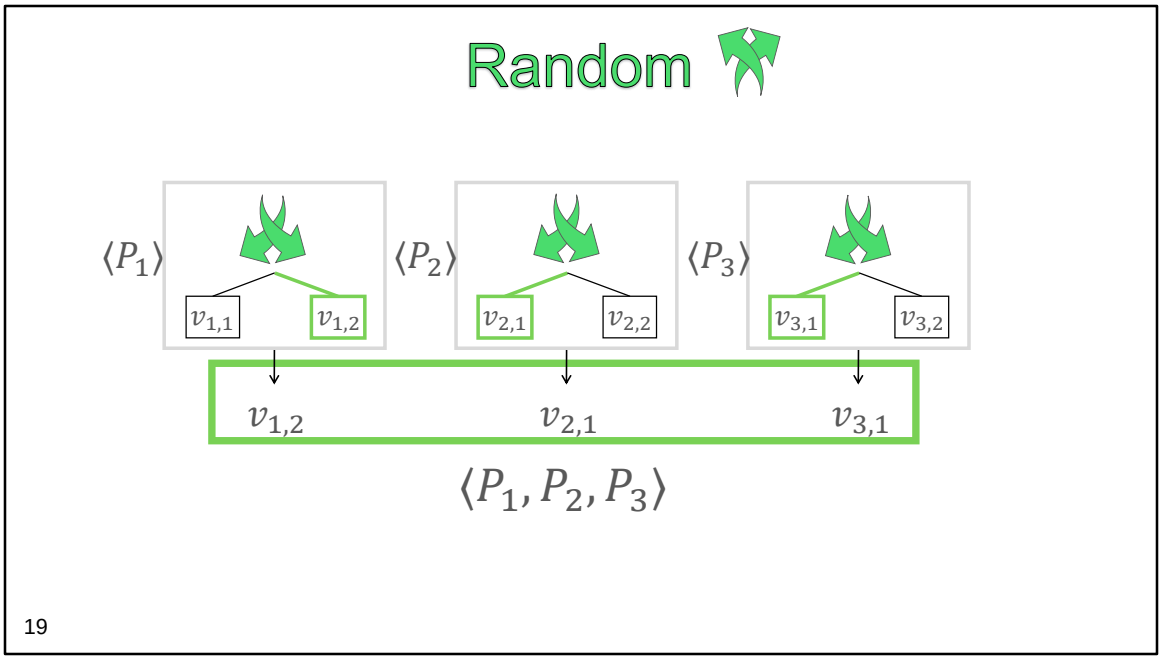
Well, it looked something like this before. Now, before modifying the population for the next generation, we ask the tuner to provide the parameters to be used for this iteration. The rest of the loop works the same, but after evaluating this new population, we return the value to the tuner as feedback for the quality of the suggested parameters. While RHEA attempts to maximize the fitness of the individuals (thus sequences that lead to the highest score), the tuner attempts to give those parameters which lead to the highest fitness improvement from one generation to the next.

Tuners

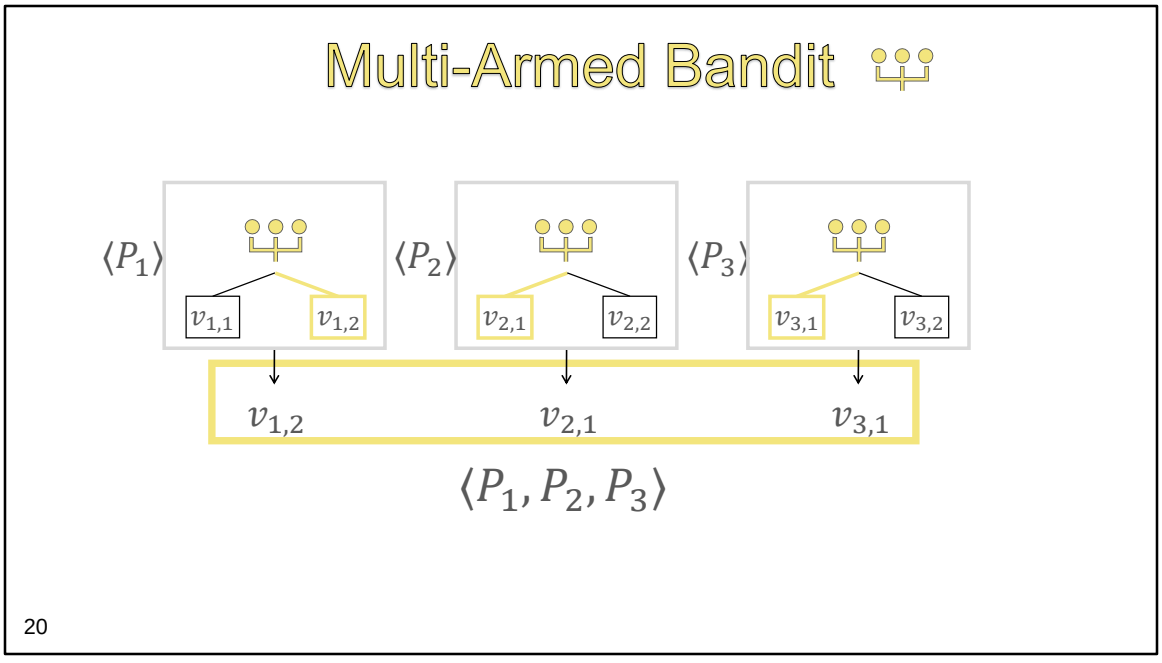


18

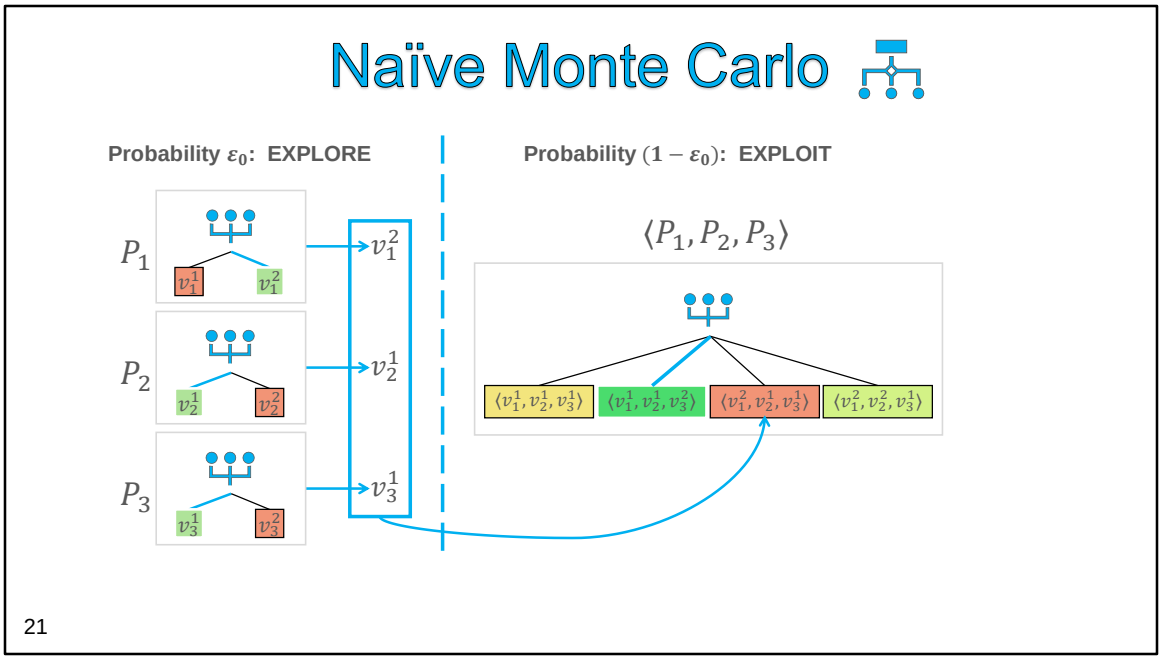
We use 5 different methods to adjust the parameters.



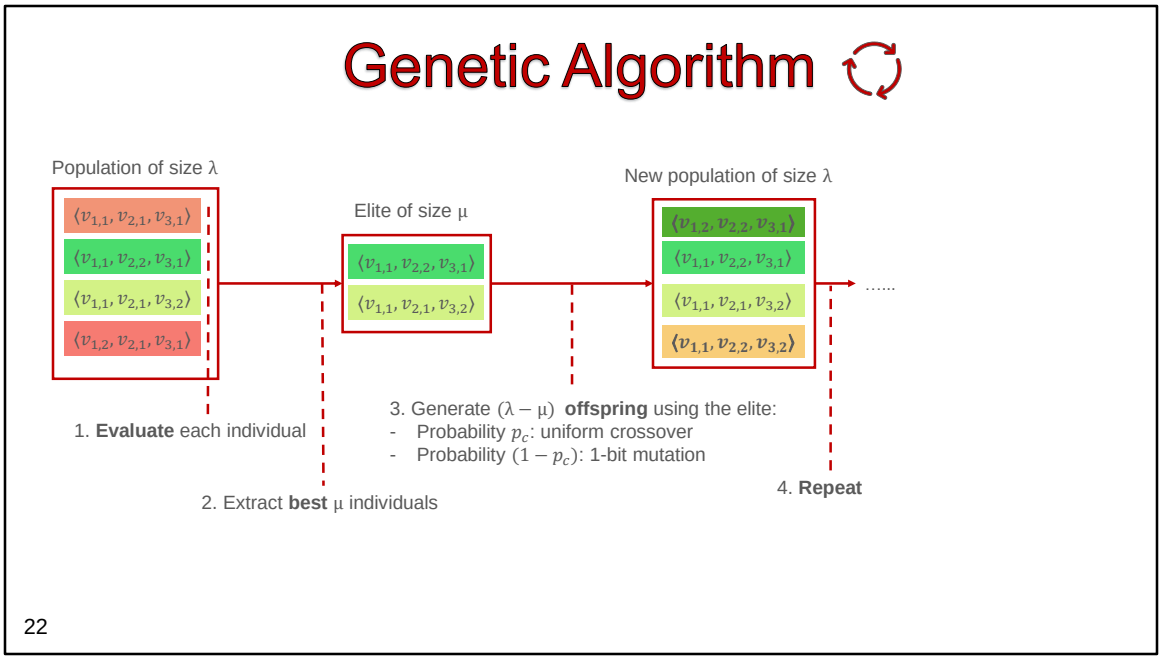
First, random. For each of the different parameters, this method randomly picks one valid value in order to form the set of parameters for an iteration. It does not use the feedback of RHEA population improvement, but we do gather those statistics for post-processing.



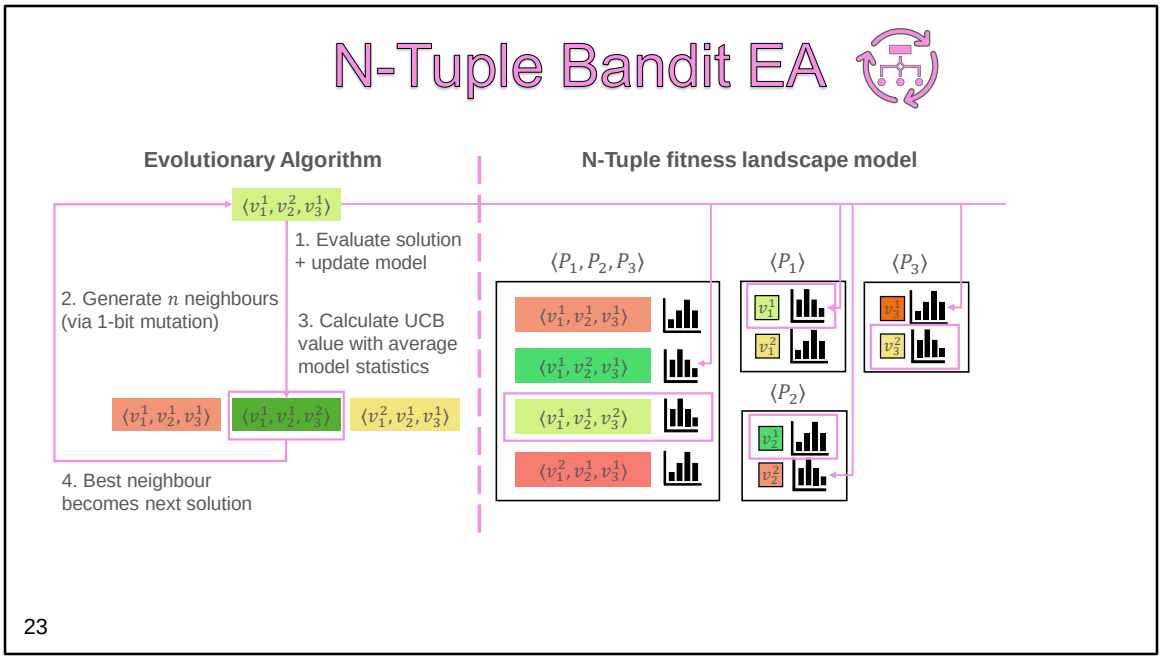
Second, a multi-armed bandit. Similarly, this method chooses a value for each of the parameters. However, this uses statistics on how many times each value for each parameter was chosen and what was the value returned, so that it balances between exploring those values least explored, and exploiting those values which appear to lead to the best results.



Third, naïve monte carlo. This builds upon the previous method (seen on the left side), which is applied with probability epsilon. More often, however, it will instead choose to use a global multi-armed bandit instead, which will look at complete sets of parameter values, using the statistics gathered through all previous samples.

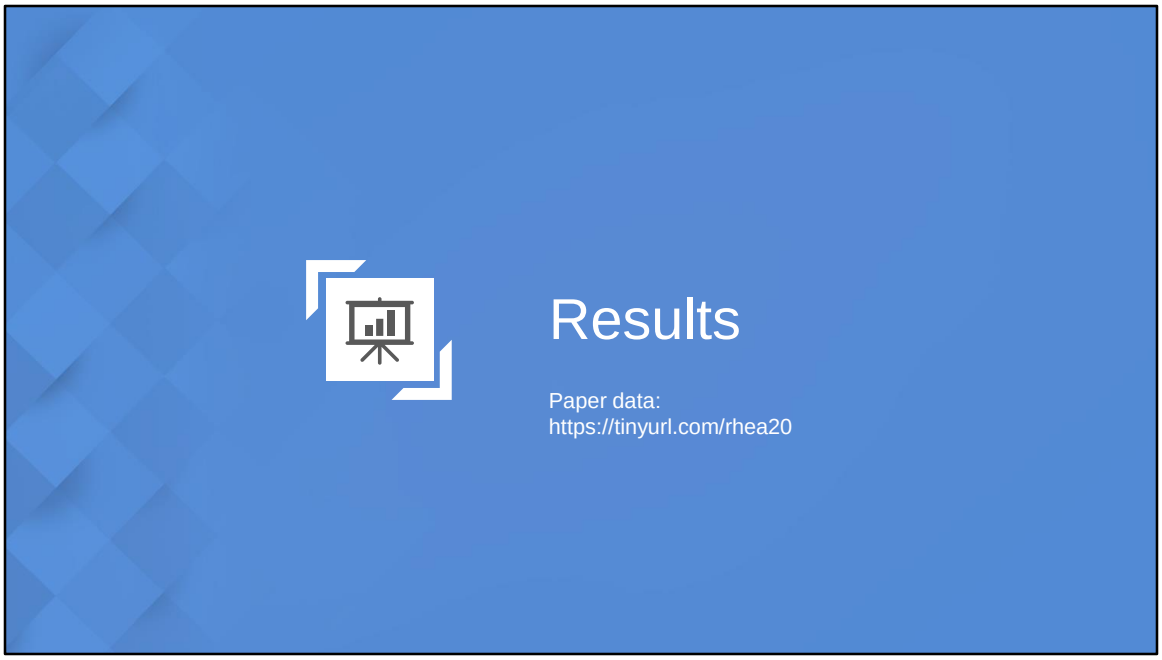


Fourth, a genetic algorithm. Here, individuals are represented by sets of parameters instead, and they are each evaluated as previously seen through one RHEA iteration. Once all individuals in the population have been evaluated, the best miu algorithms are selected, which will be used to generate offspring through uniform crossover, or through 1-bit mutation, with some probability for each method. This results in a new population and the process is repeated.

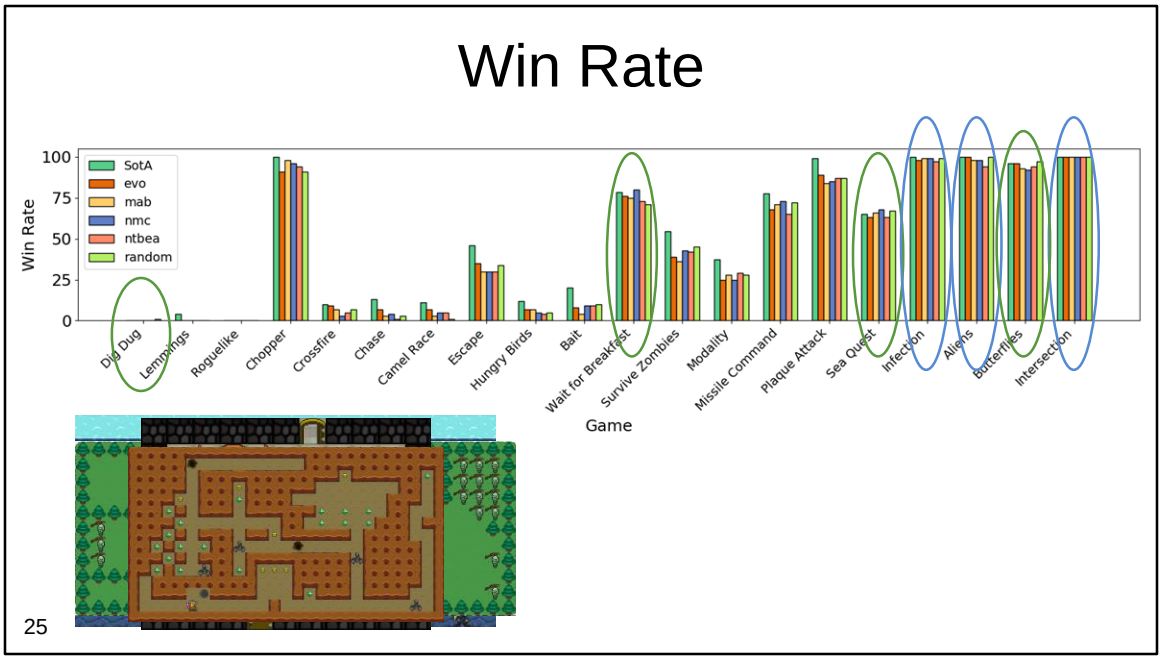


And the last tuner is the N-Tuple Bandit Evolutionary Algorithm (or NTBEA), which again uses an evolutionary algorithm which evaluates one solution at a time and uses the value returned to update an internal fitness model. For example, this individual would put one entry into the 3-tuple records with its corresponding value, and 3 other entries for each parameter 1-tuple, with its chosen parameter value and returned feedback value.

From this solution, *several* neighbours are generated through 1-bit mutation, and the statistics in the model are used to approximate the value of all of these new individuals (using a UCB value similar to the Multi-Armed Bandit and Naïve Monte Carlo tuners). The neighbour with the highest value is chosen as the next solution to be evaluated, and the process repeats.



Let’s see how these did in action! The link here includes more results, log summaries and code to obtain the plots and tables from the paper.



First, we'll look at the win rates obtained by the tuned agents. Here we see a plot with win rates in each of the games tested, one bar per tuning method, with RHEA state of the art results in green. It's important to note that these are best results obtained by any previous RHEA variant, and not a single algorithm as is our case.

The first thing to note is that the tuned agents perform very similarly to each other, with small differences (not significant). There are a few games where the results are on par with the state of the art – generally high win-rate games. But there are also some in which some of these agents perform even better. Some highlights are Butterflies and Seaquest, in which the random tuner achieves the highest performance – these are stochastic games with very dynamic environments, in which it appears to be beneficial for the agent to be switching its parameters often. A similar peak can be seen in Wait for Breakfast, a game about finding the right seat and waiting for food to be delivered: very simple in concept, but not always the easiest to get right for these general players. And we can also see first win rates in Dig Dug – a game with a large and complex environment, showcasing different navigation, puzzle and obstacle avoidance tasks. The variation introduced by the change in parameters helps the agent solve this problem, although more of such instances are needed in order to be able to analyse its performance and discover the

key to success.

Parameter Combinations (1)

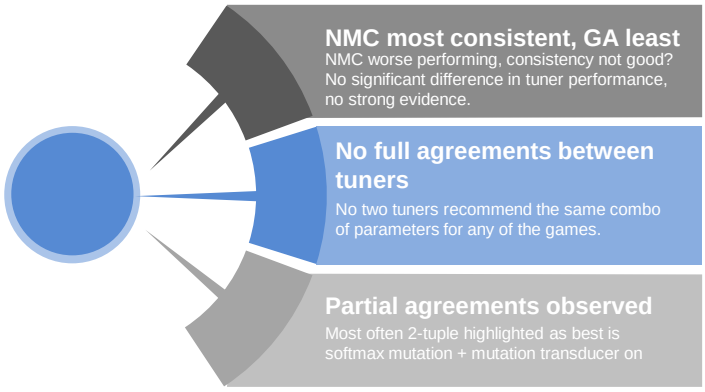
					Game:Tuner
Mutation only	Tournament	1-point	Softmax	True	*Lemmings*:NTBEA, *Crossfire*:NMC, *Aliens*:RND, *Intersection*:RND
Crossover + Mutation	Roulette	Uniform	Softmax	True	*Chopper*:RND, *Escape*:MAB, *Seaquest*:RND, *Butterflies*:EA

Most chosen combinations (5-tuples)

26

Next, we looked at what parameters were favoured by the tuning methods, as this can give further insight into what works and what doesn't in the algorithm. Showcased here are the two most chosen combinations as the best across all games, therefore the highest agreement amongst the tuners as to which parameters work best together. In both of these we observe the combination of softmax mutation with the mutation transducer enabled, which is a trend in several other games.

Parameter Combinations (2)

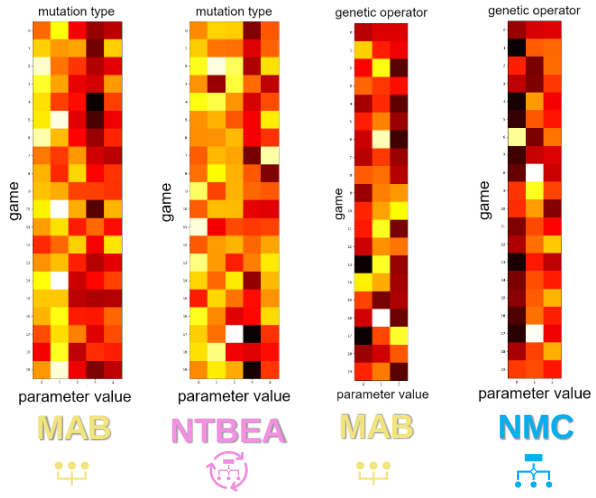


27

We measured the consistency in recommendations across all tuners, and observed Naïve Monte Carlo to be most consistent, while the genetic algorithm was the least consistent. The NMC tuner was on average worse performing, suggesting that variation is actually beneficial when higher win rates are targeted, although the difference was not significant and thus we have no strong evidence in this direction.

If we look at the games individually, no two tuners recommended the same combination of parameters for any of them, but there were some partial agreements in 2-tuples observed, as mentioned previously.

Individual Parameters (1)



RHEA configuration:

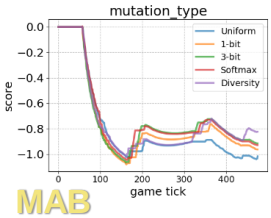
- 5 Population Size
- 10 Individual Length
- 1000 FM calls

28 Normalised count of times the parameter value was considered best

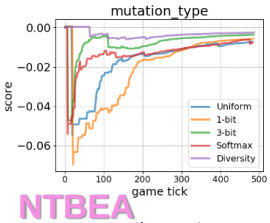
And looking at the individual parameters, there were some clear preferences in some cases for particular values. Most interestingly, we note that no tuner valued softmax mutation very highly in isolation, although it appears in several best 5-tuple recommendations. 1-bit mutation is preferred instead on average. We further note that the mutation genetic operator was preferred by most tuners, while crossover only was deemed the worst option.

Individual Parameters (2)

Wait for Breakfast

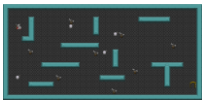


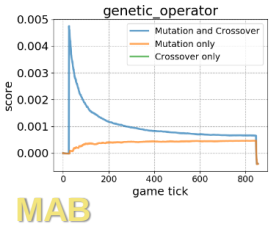
MAB



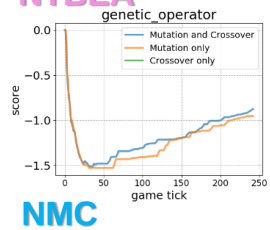
NTBEA

Crossfire





MAB



NMC

RHEA configuration:

- 5 Population Size
- 10 Individual Length
- 1000 FM calls
- Winning instances

29

And if we look at specific examples of parameter values in specific games, we can observe the difference in the tuners' thinking process, as well as an interesting dip or steep increase after only a few game ticks: this suggests that the initialisation methods could be improved for all tuners. Not all of them converge in the given time for a game either (the Naïve Monte Carlo tuner is a good example shown here), so playing longer games, with more information and more time to adapt to the specific games and scenarios could show an overall boost in performance.

To summarise, we automatically adjusted parameters for a rolling horizon evolutionary algorithm during play, with several optimisers…

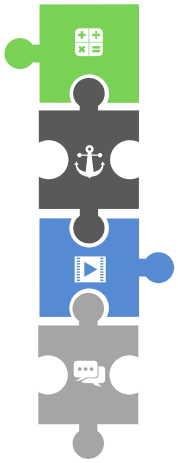
Interesting Findings

Tuned Agent Performance 01
Win rates comparable to the RHEA state of the art, surpassing it in several games.

Tuner Performance 02
Tuner performance was very similar, with RND and NTBEA standing out in a few cases

Parameter Highlights 03
Best in combination: softmax mutation + mutation transducer
Best: 1-bit mutation; mutation only (genetic operator); mutation transducer

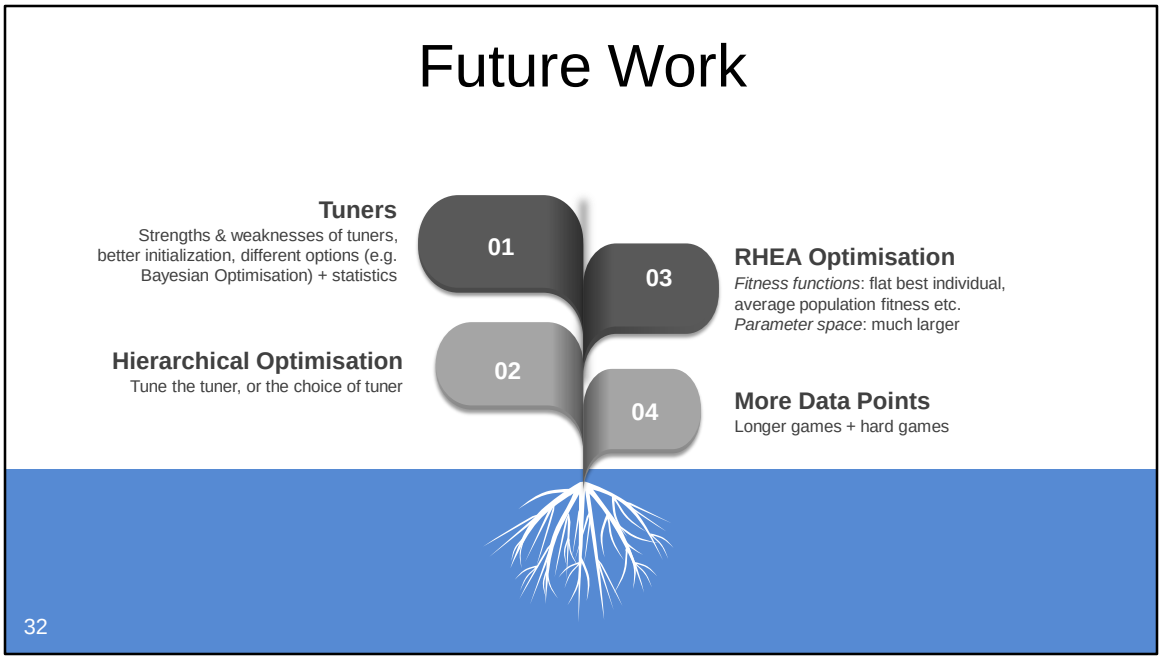
Difficult Problems 04
New win rates in very difficult problems such as *Dig Dug* and *Lemmings*



31

We've found that win rates of the tuned agents are comparable with the hand-picked RHEA state of the art, surpassing it in several games, thus more general as a single algorithm. The performance of the optimisation methods was very similar, although the random and n-tuple bandit evolutionary algorithm stood out as better; no significant differences were observed. From the parameter analysis, we noticed a combination of the softmax mutation with the mutation transducer was most beneficial in many cases. However, if dependencies are ignored, 1-bit mutation was individually better, as was only using mutation to generate individuals, with the mutation transducer enabled.

Lastly, we saw several win rates in very difficult problems such as Dig Dug and Lemmings….



… which we aim to investigate further: obtaining more data points of such winning games and analysing the resulting statistics could offer important information on how to tackle these very difficult problems; and, in general, playing longer or more complex games, with more iterations for the algorithm, could give more statistics for more accurate performance checks and possibly more significant differences in the methods used.

The tuners can further be investigated and analysed in more depth, as well as trying different options, with Bayesian Optimisation as an example kindly suggested by a reviewer. Even more so, the choice of tuner and the parameters of the tuners themselves could be optimised, for a many-level hierarchical optimisation problem!

And lastly, the fitness function for RHEA evaluations could be improved, and things such as the average or standard deviation of fitnesses in a population be considered. Not to mention that the parameter space for the algorithm is much larger than that explored here, and many more interesting insights could be given by a larger-scale optimisation.

The paper itself and full details of our methods can be found at the link attached, and you can contact me or any of the authors for any questions you might have. Thank you for watching!