Universitat Rovira i Virgili

Master's Thesis

Masters of Artificial Intelligence and Computer Security

Behavior approximation using fuzzy-genetic systems

Mark Safronov

August 10, 2025

Contents

1 Introduction

In this section we'll set up the context of a problem to be solved and briefly cover the most obvious approaches to its solution. Readers which don't require such an introduction can proceed directly to the 2.

1.1 A game solver as a scheduling problem

In 1993 in Japan Studio Gainax released a computer game called *Princess Maker 2*. The game belongs to a so-called "life simulation" genre, the player takes the role of a guardian of a young girl, making decisions that affect her upbringing and future. *Princess Maker 2* is a narrative-heavy game, with a diverse set of gameplay elements depending on player's choices, but under the plot and all the other art elements which normally constitute a computer game, lies very formal and routine core gameplay loop:

1. The player estimates the current state of the girl;
2. The player chooses the day-to-day schedule for the girl;
3. The girl "performs" the scheduled actions;
4. The game changes the state of the girl according to the actions performed.

Story-wise, the game ends after ten years of upbringing the girl. At this point the game evaluates the state reached by the girl and tells the final "fate" she got as the result. The title of the game is the hint on the ultimate goal, but depending on the final characteristics of the girl (and some special story-dependent flags we ignore in this work) game can end in more than twenty different endings.

This is essentially a stateful agent planning its behavior according to the predefined goal to be reached. Thus, the question arises: can we solve it? Given the desired ending, can we deduce the total "schedule", a full list of choices to make to get to this ending, automatically and in one go? This question lies in the core of this work.

1.2 Existing methods assessment

There are two approaches which are deceptively obvious choices to solve this problem, namely, automatic planning theory [**automaticPlanning**] and reinforcement learning [8].

1.2.1 Automatic planning theory

First, we can directly use one of the modern numeric solvers, for example, ENHSP [6] [5].

This is proven to work on small-scale problems: the code repository of the ENHSP-20 solver [1] has benchmarks with essentially 40 numeric parameters and 10 action choices.

But the very nature of the automatic planners, namely, the intent to search for the shortest schedule reaching the goal, contradicts the problem we want to solve. The Princess Maker problem dictates delayed evaluation of the goal state, we must execute a set amount of actions, and stay at the goal state at that time. More than that, as we define more rigorously in the section 2, there are no "no-operation" actions, we cannot "do nothing" to pad the actions sequence, and every action changes the state.

So, while using the automatic planners almost perfectly matches our needs, this is not the goal of this work. In addition to that, the problem of scale stays unexplored, namely, the specific Princess Maker problem is just an example of a large-scale planning problem which we want to explore in this work.

1.2.2 Reinforcement learning

Second, we can try applying the Reinforcement Learning [8] to this problem. The choice of this optimization method is deceptively obvious, because at a glance, the problem looks like a perfect match for it. We have an agent, which has a state, and this agent can perform actions which change the state. Ultimately we want the agent to reach the goal state which will give it the best reward.

The issue of scale is the main obstacle in this case.

The origin problem of solving Princess Maker 2, described in the previous subsection, assumes 25 actions over a state space of 50 numeric characteristics each one having values between 0 and 500. Just enumerating the possible states of the agent caused by these actions in the

[1]https://github.com/hstairs/enhsp/tree/enhsp-20

state space of such a size is an intractable problem, which we will rigorously show in 4. Just the rewards table for this has a size of 25×500^{50}, which is already practically intractable.

Moreover, the most important problem is the length of the process. Following the base example of *Princess Maker 2* gameplay, player makes 3 choices per virtual "month", and the game spans 10 "years", so the search space is a tree $3 \times 12 \times 10 = 360$ levels deep.

2 Formal problem statement

As discussed in 1.2, formally we have a choice of whether to treat this as a planning problem or a control problem. Despite Reinforcement learning not being fit for our cause, we will still approach our origin problem as a control problem.

2.1 Actor behavior as a control problem

Assuming we have a character described as a set of numeric characteristics

$$\mathbf{x} \in \mathbb{Z}^n \tag{1}$$

we have a set of possible actions

$$A = \{a_1, a_2, \dots, a_m\} \tag{2}$$

which collectively form a transfer function

$$f(\mathbf{x}, a) = \mathbf{x}' \tag{3}$$

To describe the desired outcome, we first declare a fitness function mapping the state to a numerical value:

$$\Phi : \mathbf{x}' \to \mathbb{R} \tag{4}$$

a goal fitness value

$$\mathbf{G} \in \mathbb{R} \tag{5}$$

and a planning horizon

$$T \in \mathbb{Z} \tag{6}$$

We want to get an ordered actions sequence of length T which will lead $\mathbf{x}$ to some $\mathbf{x}^*$:

$$\mathbf{a} \in A^T, x_o = \mathbf{x} : \bigodot_{i=1}^{T} f(x, a_i) = \mathbf{x}^* \tag{7}$$

(where $\bigodot$ is a fold operator)
such as:

$$\Phi(\mathbf{x}^*) > \mathbf{G} \tag{8}$$

The transfer function f is assumed to be completely determined, and the whole process being non-stochastic. This is a significant restriction which cannot be lifted for the proposed solution to work.

2.2 Dimensionality explosion stemming from the original context

We assume a fixed-length trajectory of actions, each of which transforms the state of the system according to a known deterministic transfer function (3).

This means two restrictions:

1. No-operation actions are prohibited, each step must result in a meaningful state transformation, reflecting the irreversible nature of time.
2. Goal state must still be in effect at the step T.

While the agent cannot avoid taking actions — and hence cannot avoid changes to the system — it is allowed to evaluate its progress toward the goal at every intermediate state. In this sense, the problem is not a pure planning task but an episode-based control problem with delayed evaluation.

In this work we'll focus specifically on the cases which lead to combinatorial explosion for classical solutions, that is, when we have sufficiently large amount of characteristics, actions to choose from and most importantly, very large planning horizon:

$$n > 50 \tag{9}$$

$$m > 20 \tag{10}$$

$$T > 360 \tag{11}$$

The origin *Princess Maker* problem is the lower edge of the cases we are interested in.

With these restrictions in place, a need in an heuristic arises to perform efficient search in the state space, as its size becomes unrealistically large.

3 Proposed solution approach

Classical reinforcement learning methods become intractable in this domain due to the high dimensionality of the state space, large action set, and long planning horizon. Moreover, the inability to halt or take neutral actions further exacerbates the combinatorial explosion of the trajectory space.

To address this, we introduce a heuristic dimensionality reduction via the concept of inclinations — latent behavioral parameters — and model the behavior policy as a fuzzy controller which maps the current state and inclinations to a concrete action.

This parametrization constrains the space of possible behaviors, making the optimization tractable. Instead of learning or searching over action sequences directly, we perform optimization in the significantly smaller space of inclinations, evaluating the final outcome after T steps. The resulting problem becomes an offline, black-box control task — suitable for evolutionary algorithms, rather than classical RL methods.

3.1 Using domain knowledge to reduce dimensionality

In this work we evaluate an approach which is defined as follows.

Let's assume that we can segment the set of possible actions to clusters with the following particularities:

1. actions in the same cluster lead to "similar" changes in the character state $\mathbf{x}$.

2. the cluster as a whole can be described symbolically

In this case we can synthesize a set of numeric characteristics which we'll call "inclinations":

$$\mathbf{I} \in \mathbb{Z}^q \tag{12}$$

$$q << n \tag{13}$$

From this, we can define a set of fuzzy rules[4] mapping the inclinations to action choices:

1. if an inclination I_i has a fuzzy value V_I,
2. and the current state $\mathbf{x}$ has fuzzy values V_i^x
3. then P_a, the priority of an action a, is a fuzzy set V_A.

After the defuzzification of all the inferred fuzzy values P_a we select an action with the highest priority.

The selection and design of fuzzy rules is a critical aspect of this approach. In this thesis, the fuzzy rule base is constructed manually, leveraging domain knowledge to define the mapping from inclinations to action priorities. Future research may investigate automated methods for generating fuzzy rules, such as clustering or machine learning techniques, to further improve scalability and reduce manual effort.

While it is theoretically possible to define fuzzy rules that map every possible inclination vector $\mathbf{I}$ or even every state $\mathbf{x}$ to action priorities, such exhaustive rule sets would quickly become infeasible due to combinatorial growth. This reinforces the importance of dimensionality reduction and clustering in making the fuzzy-genetic approach tractable for high-dimensional planning problems.

The assumption which we explore among others in this work is the practical possibility to write a coherent set of fuzzy rules which will be clustered around the clusters of actions, and each inclination will tend to map to its own cluster of actions.

Now, using such a fuzzy controller $\xi(I, \mathbf{x})$ we can construct the goal function:

$$g(I, \mathbf{x}) = \bigodot_{i=1}^{T} f(x, \xi(I, x_i)) \tag{14}$$

the above formula being subject to improvements in expressiveness, the main point of which being the fuzzy controller $\xi(I, x_i)$ selecting the action to perform on the step i according to the inclinations and (ideally) the current state x_i.

The argument $\mathbf{x}$ is essentially a constant for both (7) and (14). As the transfer function f is non-stochastic, $\mathbf{I}$ uniquely maps to the actions sequence $\mathbf{a}$. Thus, given (13), we effectively performed dimensionality reduction on the original problem.

We can find $\arg\max(g)$ now using an appropriate optimization method. For this work, because of a strong biosocial analogies a genetic algorithm[2] was chosen, with the vector of inclinations $\mathbf{I}$ as a chromosome.

3.2 Hypothesis

The hypothesis explored in this work is that the combination of assumptions described above constructs an heuristic which allows finding a locally optimal solution in a polynomial time.

3.3 Objectives

The objective of this thesis is to investigate whether the stated hypothesis is true. That is, whether a fuzzy-genetic heuristic can effectively solve high-dimensional deterministic planning problems through dimensionality reduction and symbolic reasoning.

In particular, we aim to:

1. Formalize the problem as an optimization task.
2. Implement a working solver.
3. Evaluate the performance of the solver on a set of test cases of increasing complexity.
4. Analyze the results to draw conclusions about the effectiveness of the approach.

4 Methodology

In this section we will discuss the theory which this work is build upon, namely, fuzzy logic [4] and evolutionary algorithms [2].

4.1 Encoding domain knowledge of actions as a fuzzy controller

Normally the Fuzzy logic is being explained from the fuzzy set theory by L. Zadeh [**zadehFSbase**], but for this particular work the most important part of the fuzzy logic is the fuzzy rules for the fuzzy controller so it's more beneficial to start with them.

In the scope of the FL it is possible to express the domain knowledge in the form of symbolic rules, with the general form as follows:

```
If (input variable A) has a (fuzzy value Fa)
 then (output variable B) has a (fuzzy value Fb)
```

For example, for our particular problem and solution method:

```
If InclinationAggressiveness is High
 then DuelingClassesPriority is High
```

This rules format depends on the concept of the Fuzzy Variable, which is a combination of four major parts:

1. Name
2. Range of "strict" values
3. "strict" value itself
4. Set of fuzzy sets describing the possible fuzzy values of this variable

The concept of Fuzzy Variable, in turn, depends on the concept of a fuzzy value, which is a combination of two major parts:

1. Name

2. Membership function

A fuzzy controller is an algorithm which performs three large steps:

1. Convert all the strict values of the input fuzzy variables into the fuzzy values
2. Evaluate all the fuzzy rules, obtaining the fuzzy values of the output fuzzy variables
3. Apply the special *defuzzification* operation to the output fuzzy variables, obtaining their strict values.

In our system, we're going to have the vector of inclinations and the current state of the specimen as input variables for the controller, and have the priorities of actions as the output variables. This will allow us to imitate the process of "decision making" of the specimen to choose the next action to perform.

The major benefit and the core reason for the fuzzy controller is the ability to encode the domain knowledge in a limited set of rules which will be formally processed.

Compared to, for example, some of the reinforcement learning methods, we don't need to specify the full table of rewards for every possible action-state combination. It is enough to specify one rule for every available action and the controller will already become fully functional. With some configuration of rules it's possible to write even less of them.

This allows to simplify the implementation of the solver, because one of the main weaknesses of the proposed solution is writing the fuzzy rules by hand.

The second benefit of using the fuzzy controller for decision making is that it can be applied without major changes to non-deterministic, stochastic environment, for example, if the actions would be allowed to make randomized changes to the specimen's state, that is, if the transfer function would not be pure. It opens up the possibilities to explore this topic further in the later works.

4.2 Control feedback loop as a fitness function

A single trajectory in the action space is explored using the following process.

1. We start with the initial state $\mathbf{x}_0$ and the given set of inclinations $\mathbf{I}^k$
2. We evaluate both $\mathbf{x}_0$ and $\mathbf{I}^k$ with the preconfigured fuzzy controller
3. The defuzzified output of the controller is the set of priorities for all the actions. We pick the action with the highest priority. Tiebreaker is the position of the action in the list.
4. Action is executed and if we haven't made T actions yet we return to the step 2
5. After T executed actions we apply the goal conditions predicate $\Phi(\mathbf{x}*)$ and calculate the fitness based on that.

It is important to understand that the state of a specimen is a transient value, used only for calculations of the final fitness after T iterations. The solution we seek is fully encoded in the inclinations vector $\mathbf{I}$, which stays unchanged for the entirety of the control loop.

4.3 Global optimization using an evolutionary algorithm

Strong biosocial analogies and the configuration of the control loop from 4.2 suggest us to use the evolutionary algorithms [**evolutionary**] for optimization. This is what would be used in this work. However, in principle, any algorithm which is able to use the concept of a fitness function would be applicable here.

Evolutionary algorithms can be explained with an example of the so-called Simple Genetic Algorithm [**pagmo:sga**].

SGA operates on the set of specimens, each one being a single option in the search space to explore. A specimen is classically a list of characters, which is called literally a genome. The whole set of specimens is called a population.

In our case, a specimen would be a list of inclination values.

Every specimen in a population is evaluated using the fitness function, producing a fitness value.

Then, a *selection operator* is applied, choosing a subset of the population. For example, our selection operator may be choosing the top 50% of the population by their fitness value.

After the selection, we apply the *crossover operator* to the pairs of selected specimens' genomes. The classical crossover operator picks a single place inside both of the genomes and then swaps the resulting halves between them. For example, a genome 'aaaa000' and a genome '1111bbb' after the crossover at point 5 become 'aaaabbb' and '1111000'.

After the crossover we apply the *mutation operator* to all of the selected genomes. Mutation changes with some low probability individual genes in the genomes at random. For example, we can have a mutation operator which has 0.01 probability of flipping a gene in the genome from 'a' to 'b' and *vice versa*. Then, we have 0.002 probability of a specimen with a genome 'aaabb' turning into 'ababb'.

After the crossover and mutation, we finally apply the *replacement* which forms the new population for the next generation and the next round of evolution. For example, we can use a so-called $(\mu + \lambda)$-evolution strategy: calculate the fitness for all the new genomes and then pick s ones with the best fitness from both the old genomes and new ones, where s is the target population size. The size of the population is being kept constant for the classical genetic algorithms, the role of the replacement operator is specifically to enforce that.

In the approach described in this work, the fitness function is the control loop described in the previous section **??**. The genome of the specimen is the vector of inclinations. And due to the choice of the specific library for the implementation of the evolutionary algorithms we have a wide selection of them, which means, we can explore different options starting from the Simple Genetic Algorithm and continuing with more complicated options.

The library Pagmo [1] includes a lot of already implemented different evolutionary algorithms apart from the simple genetic algorithm so it enables us easier exploration of possibilities in optimizing the full solver.

5 Implementation

The technical implementation of the method is performed in C++ [7] using the libraries FuzzyLite [3] and Pagmo [1]

5.1 Choice of a C++ language as foundation

As the root problem of this work is the problem of scale, it has been decided that we trade comfort of experimentation for pure processing power.

Contemporary C++, starting with the standard version 20, allows for a very high-level code as readable as a natural language. It also has enough libraries written for solving different scientific problems for us to not need to implement neither fuzzy controller nor the evolutionary algorithms by ourselves.

In talking on choice of the language for the implementation we cannot avoid comparisons with Python, assumed leader and language of choice for scientific experiments. As has been stated above, it has been conscious decision to trade the ability to make rapid changes in the code, especially the ability to run Jupyter notebooks, for the raw processing power. This is because the C++20 and later is expressive enough to be as readable as Python sans some of the required syntax boilerplate, and in reality the most painful part of choosing C++ is building the program to be cross-platform, as Python programs are, and doing that with the code which uses third-party libraries is a nontrivial implementation problem.

5.2 Fuzzylite library for the fuzzy controller implementation

This work turned out to be more or less an assessment of usefulness of the `fuzzylite` [3] C++ library in addition to the main goal. While being fully open sourced with a non-restrictive license terms, actually adding it to an existing C++ program is a task certainly not feasible for an arbitrary computer scientist not being the seasoned software engineer at the same time. Which is a shame, as it offers a straightforward idiomatic API which allows expressing the algorithms in a readable format.

`fuzzylite` also provides a domain-specific language for specifying the fuzzy controller, which allows us to describe this part of the algorithm in a language more expressive than the raw C++ function calls.

5.3 Pagmo library for evolutionary computations

6 Experiments and Results

In the scope of this work we'll use the original Princess Maker 2 problem but segmented in four different problems of increasing scale.

6.1 Trivial case

2 characteristics, 4 mutually exclusive actions, 3 steps.

This scenario represents a trivial case, with only 4^3 possible action sequences—a total of 64. The small state space allows for exhaustive enumeration and manual verification of results. This case serves to validate the correctness of the implementation and the fuzzy controller, as the system's behavior can be easily traced and analyzed by hand.

6.2 Base control case

4 characteristics, 12 actions, 100 steps.

This case is the base case, as it introduces enough complexity to test the proposed approach and at the same time compare it with classical approaches.

A decision tree of the size 12^{100} is already too large to be completely enumerated.

However, with 4 characteristics and 12 actions, the problem is still well within the range where Reinforcement Learning methods—especially those using function approximation—can be applied efficiently. The planning horizon of 100 steps is long enough to be non-trivial, but does not pose significant challenges for standard RL algorithms.

6.3 Origin case

The complete Princess Maker 2 case is a problem with 50 numeric characteristics of a character and 25 actions to choose from, with a planning horizon of 360 steps.

This case is an attempt to directly solve the original problem which started this work. It will be used as a benchmark for the proposed solution on a real-world problem.

7 Discussion

Using the fuzzy controller came out more complicated than it could be seen from the theory alone. While the target of the work was reducing the search state space, the amount of parameters in the fuzzy controller exploded the hyperparameters space instead, as by different configuration of the fuzzy rules and fuzzy variables we can change the behavior of the specimen.

If we exclude the current state of the specimen from the fuzzy rules, we trivialize the trajectories, reducing them to repetition of the same action T times. While this does not simplify the optimization step, as it is assumed that though (13), q is still large enough for the bruteforce enumeration to be intractable, it leaves us with action sequences intuitively unfit as solutions for any realistic nontrivial goal states.

If we setup the fuzzy action-prioritizing rules in such a way that they would indeed use the current state of the specimen, we do turn the problem into the control one with non-trivial solutions, but at the same time we end up having to specify not only at least one rule per each action priority, but also at least one rule per each *term* per *each input variable*, which starts competing with the complexity of the problem we are trying to solve by this method in the first place.

8 Conclusions and Future Work

First, let us make a conclusion tangential to the origin problem but related to the chosen theoretical toolset. While the fuzzy controller can indeed be seen as a tool to formalize the decision making using the expert knowledge in the given domain, it is too complicated mechanism by itself to help reducing the complexity of the problem it's solving. One should treat it not as a tool which helps make complicated problems simpler but which, hopefully, makes unsolvable ones solvable at all, as proper configuration of the fuzzy controller is already a problem in itself.

Despite the context of the problem being a computer game, the problem itself is a general one, and the proposed approach can be applied to any high-dimensional deterministic planning problem. This constitutes the core value of this work.

In the span of this work, only three distinct cases were explored, and the more thorough exploration of the parameter space is left for a dissertation-level research.

References

[1] Francesco Biscani and Dario Izzo. "A parallel global multiobjective framework for optimization: pagmo". In: *Journal of Open Source Software* 5.53 (2020), p. 2338. DOI: 10.21105/joss.02338. URL: https://doi.org/10.21105/joss.02338.

[2] Melanie Mitchell. *An Introduction to Genetic Algorithms*. Cambridge, MA: MIT Press, 1999. ISBN: 9780262631853.

[3] Juan Rada-Vilela. *The FuzzyLite Libraries for Fuzzy Logic Control*. 2018. URL: https://fuzzylite.com.

[4] S. Kumar Ray. *Soft Computing and Its Applications, Volume II*. Boca Raton, FL: CRC Press, 2014. ISBN: 978-1-4822-5793-9.

[5] Enrico Scala et al. “Interval-Based Relaxation for General Numeric Planning”. In: *European Conference on Artificial Intelligence*. 2016. URL: https://api.semanticscholar.org/CorpusID:27984436.

[6] Enrico Scala et al. “Subgoaling Techniques for Satisficing and Optimal Numeric Planning”. In: *Journal of Artificial Intelligence Research* 68 (Aug. 10, 2020), pp. 691–752. DOI: 10.1613/JAIR.1.11875. URL: https://www.jair.org/index.php/jair/article/view/11875.

[7] Bjarne Stroustrup. *Programming: Principles and Practice Using C++, 3rd Edition*. 3rd ed. Available online. Addison-Wesley Professional, 2024. ISBN: 978-0-13-830868-1. URL: https://www.informit.com/store/programming-principles-and-practice-using-c-plus-plus-9780138308681.

[8] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. 2nd ed. Available online. MIT Press, 2018. ISBN: 978-0-262-03924-6. URL: http://incompleteideas.net/book/the-book-2nd.html.