Fuzzy-Genetic Hybrid Models for Large-Horizon Deterministic Planning

Final Master's Project

Mark Safrónov

10th September 2025

Princess Maker 2 — where it all started

Princess Maker 2 — scale

22 numeric attributes

Princess Maker 2 — scale

25 actions

each changes 2+ attributes

Princess Maker 2 — scale

25 actions

each changes 2+ attributes

Princess Maker 2 — endings

61 ending each having its own requirements to meet

Example: Queen ending

Stats Required

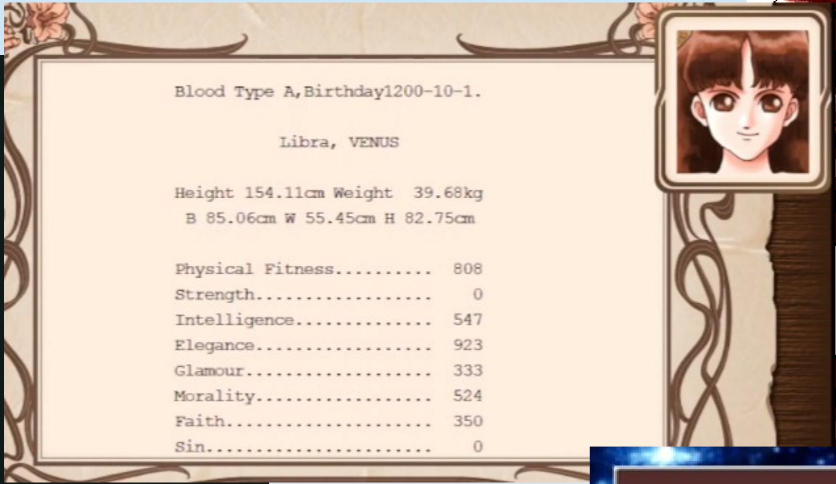
Refinement/Elegance: 800+

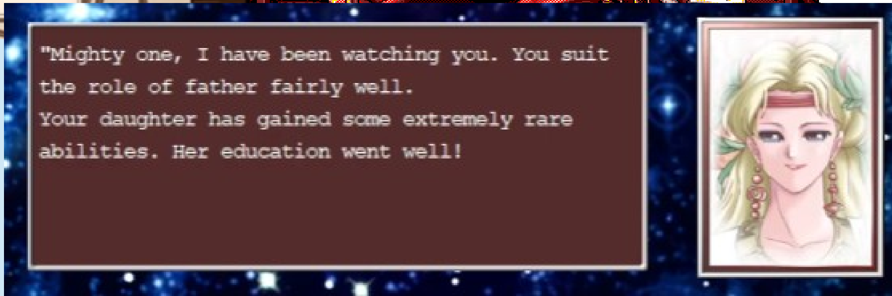
One of these stats should be 421+:
Fighter Reputation
Magical Reputation
Social Reputation
Housework Reputation

Art must NOT be her highest acquired skill stat

The gap between highest and lowest reputation should be below 50.

*She must **NOT** be eligible for a Sinful Ending (the requirements for the Sinful endings are Morality below 30 and Sin 250+ or alternatively, Morality 0 and Sin 100+)*





Can we solve that?

From the game to a CS problem

Problem: given an ending we want, can we deduce a list of actions to reach this ending?

From the game to a CS problem

One-page definition of the problem

Deterministic planning problem with the fixed-length actions sequence

2.1 Actor behavior as a control problem

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

$$\mathbf{x} \in \mathbb{Z}^n \tag{2.1}$$

we have a set of possible actions

$$A = \{a_1, a_2, \ldots, a_m\} \tag{2.2}$$

which collectively form a transfer function

$$f(\mathbf{x}, a) = \mathbf{x}' \tag{2.3}$$

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

$$\Phi : \mathbf{x}' \to \mathbb{R} \tag{2.4}$$

a goal fitness value

$$\mathbf{G} \in \mathbb{R} \tag{2.5}$$

and a planning horizon

$$T \in \mathbb{Z} \tag{2.6}$$

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

$$\mathbf{a} \in A^T, x_o = \mathbf{x} : \bigodot_{i=1}^{T} f(x, a_i) = \mathbf{x}^* \tag{2.7}$$

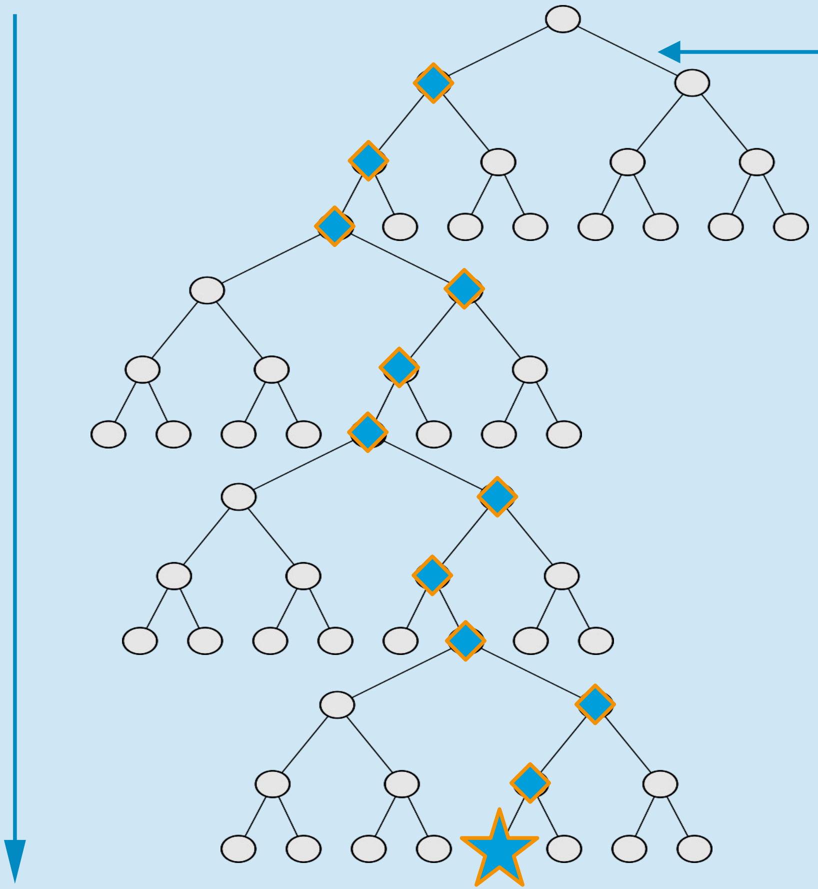
(where $\bigodot$ is a fold operator)

such as:

$$\Phi(\mathbf{x}^*) > \mathbf{G} \tag{2.8}$$

The problem of scale

Depth of the tree is **360+** steps, depending on how do we interpret scheduling



Every fan-out is **25 options** not 2

Bruteforce enumeration is intractable, what are our other options?

Option 1: Automatic planning

Courtesy of Dr. Hatem Abdelatiff.

There exist ready-made solutions based on STRIPS formalism in the PDDL syntax.
Solver like ENHSP can work with numeric states and goals.

It will not work for us because **we must have fixed length plans** — defining a cost function to enforce this is nontrivial if possible at all.

Option 2: Reinforcement learning

Courtesy of Dr. Jordi Duch.

«Stateful actor learns to perform sequences of state-changing actions to reach predefined goals»

It will not work for us because of a **scale and delayed evaluation** — after 360...1200th step the signal will be negligible.

Treat the problem as a control problem

What if I treat every trajectory as a control problem.

Then I need only to invent a way to influence the decisions in that single trajectory, and optimise on this «way»

2.1 Actor behavior as a control problem

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

$$\mathbf{x} \in \mathbb{Z}^n \tag{2.1}$$

we have a set of possible actions

$$A = \{a_1, a_2, \dots, a_m\} \tag{2.2}$$

which collectively form a transfer function

$$f(\mathbf{x}, a) = \mathbf{x}' \tag{2.3}$$

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

$$\Phi : \mathbf{x}' \to \mathbb{R} \tag{2.4}$$

a goal fitness value

$$\mathbf{G} \in \mathbb{R} \tag{2.5}$$

and a planning horizon

$$T \in \mathbb{Z} \tag{2.6}$$

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

$$\mathbf{a} \in A^T, x_o = \mathbf{x} : \bigodot_{i=1}^{T} f(x, a_i) = \mathbf{x}^* \tag{2.7}$$

(where $\bigodot$ is a fold operator)

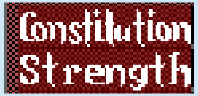
such as:

$$\Phi(\mathbf{x}^*) > \mathbf{G} \tag{2.8}$$

Clustering using domain knowledge

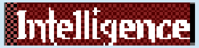
«Inclinations»

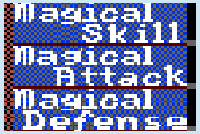
Fighting



Lumberjack, Combat classes, etc

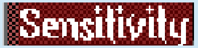
Magic





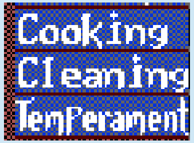
Magic classes, Graveyard etc

Artistry



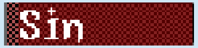
Dance classes, Tutoring etc

Housekeeping



Innkeeping, Theology etc

Sinfulness



Cabaret etc

Single trajectory

5 inclination values + current state at step *n* controls the priorities of actions

↓

Selected highest-priority action changes the current state

↓

Repeating T times gives us a trajectory and a final state

↓

Evaluate the final state (whether we are at a goal)

Optimization

5 inclination values effectively define the final fitness

We have a pure function to *argmin*

Chosen approaches

Inclinations + current state must map to action priorities: **fuzzy logic**

We must optimize inclinations to minimize fitness: **evolutionary computations**

(*minimize* not *maximize* fitness is a technical detail)

Fuzzy logic

«low»: 0.15, «high»: 0.65

«low»: 0.12, «high»: 0.67

```
rule: if InclinationFighting is high   and strength is low      then Lumberjack is high
rule: if InclinationArtistry is high   and refinement is low    then Lumberjack is low
```

Lumberjack priority «high»: 0.1, «low»: 0.1

Lumberjack priority: **0.11**
(just an example, depends on the defuzzification)

Evolutionary computations

5 inclinations: vector (I1, I2, I3, I4, I5) — «**genome**»

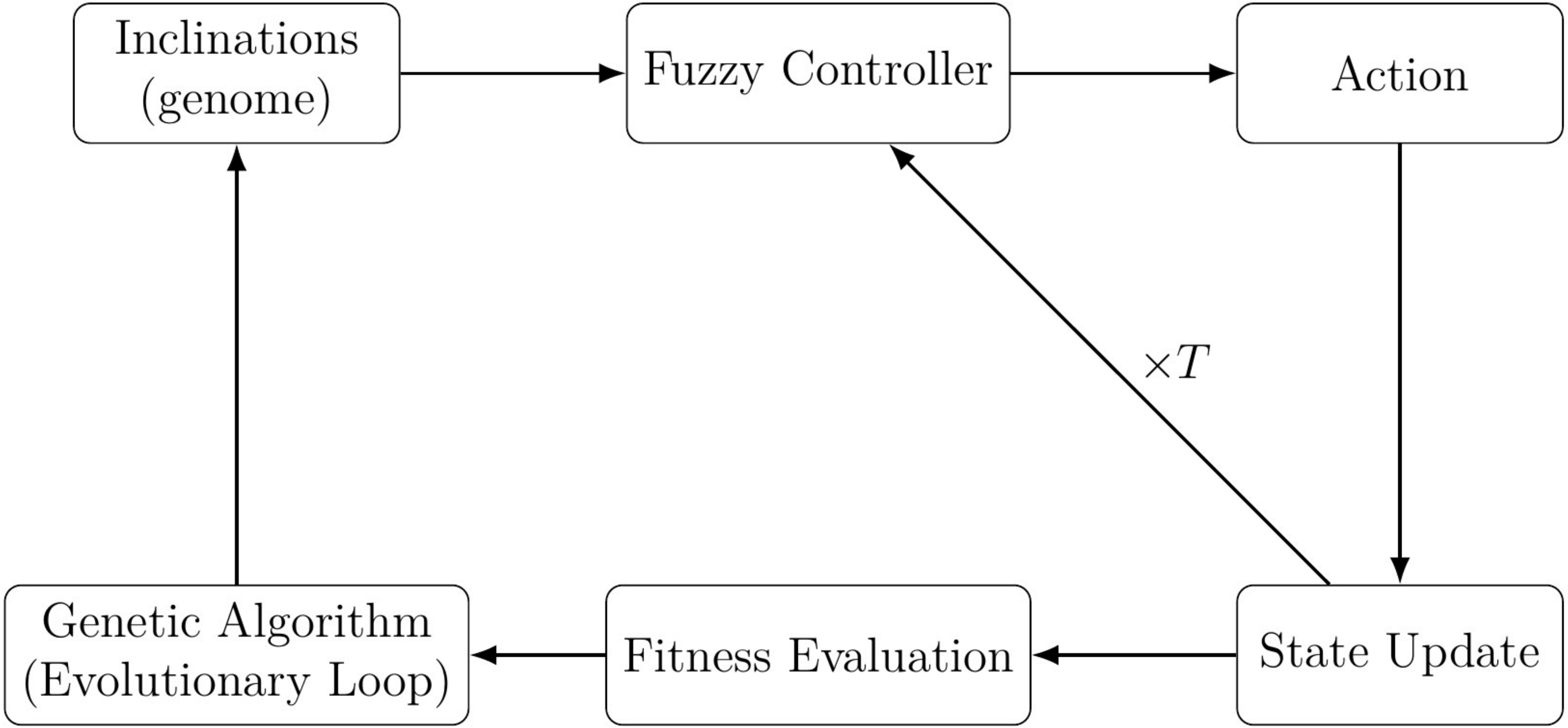
Generate N random genomes — «**population**»

For every genome, calculate their fitness.

Apply special evolution-inspired «**crossover**», «**mutation**», «**selection**» and «**replacement**» operators to the population

Repeat. If your selection and replacement are sound, computation converges.

Control flow scheme



GOALS

1. Build a proof-of-concept
2. Ensure it actually produces solutions (fitness = 0)
3. Estimate performance
4. Summarize potential problems and possible future research

Baseline experiment setup

Number of character attributes: 21

Number of actions: 25

Length of actions sequence: 1200

Complex goal predicate based on the actual game («High General ending»)

Result:

```
Stats changes:
str: +0, con: +0, int: +507,
ref: +0, cha: +0, mor: +102,
fai: +612, sin: -204, sen: -588
cs: +330, ca: +165, cd: +165,
ms: +0, ma: +0, md: +408,
dec: +0, art: +0, elo: +0,
coo: +294, cle: +294, tem: +294
Simulation Result:
Fitness: 0
```

RESULTS

Linear time on T, depends only on the complexity of the goal predicate

```
double fitness(const Stats& stats)
{
  const int fighter_reputation = std::get<0>(stats) + std::get<1>(stats)
  + std::get<9>(stats) + std::get<10>(stats) + std::get<11>(stats);

  // if sensitivity is higher than intelligence or faith,
  // return an absolute instant loss
  if (std::get<8>(stats) >= std::get<2>(stats)
    || std::get<8>(stats) >= std::get<6>(stats)) {
    return std::numeric_limits<double>::max();
  }

  double fitness_value = 0.0;

  // add penalty for intelligence below 500
  if (std::get<2>(stats) < 500) {
    fitness_value += (500 - std::get<2>(stats));
  }

  // add penalty for morality below 30
  if (std::get<5>(stats) < 30) {
    fitness_value += (30 - std::get<5>(stats));
  }

  // add penalty for faith below 300
  if (std::get<6>(stats) < 300) {
    fitness_value += (300 - std::get<6>(stats));
  }

  // add penalty for fighter reputation below 421
  if (fighter_reputation < 421) {
    fitness_value += (421 - fighter_reputation);
  }

  return fitness_value;
}
```

```
Stats changes:
str: +0, con: +0, int: +507,
ref: +0, cha: +0, mor: +102,
fai: +612, sin: -204, sen: -588
cs: +330, ca: +165, cd: +165,
ms: +0, ma: +0, md: +408,
dec: +0, art: +0, elo: +0,
coo: +294, cle: +294, tem: +294
Simulation Result:
Fitness: 0
```

Requirements clauses amount	Runtime on a reference machine
full "General" ending goal	12 m 30 s
no reputation check	6 m 30 s
no reputation, no faith checks	5 m 50 s
no reputation, no faith, no morality checks	6 m 10 s
no attribute checks, only the predicate	3 m 20 s

RESULTS

Result coming out of the solver is a vector of inclinations (5 real numbers in our setup).

Because the problem is formulated completely deterministic, we can effectively decode the sequence of actions out of these inclination values.

As such, our proof-of-concept deduces action sequences of fitness 0 → our goals are reached.

1. TheologyClass x 383
2. FencingClass x 106
3. FightingClass x 106
4. TheologyClass - FencingClass - FightingClass x 24
5. TheologyClass
6. FencingClass - FightingClass x 2
7. TheologyClass - FencingClass - FightingClass x 33
8. Housework x 294
9. Church x 63
10. Church - ScienceClass x 3
11. Church
12. Church - ScienceClass x 6
13. Church
14. Church - ScienceClass x 7
15. Church
16. Church - ScienceClass x 6
17. Church
18. Church - ScienceClass x 7
19. Church
20. Church - ScienceClass x 4
21. Church

Conclusions

1) A generic problem has been formulated and a proof-of-concept solver has been built for it.
2) We successfully utilized domain knowledge to reduce dimensionality of the original problem.
3) On the given instance of a problem the solver reaches linear time over length of action sequence T. Complexity of the goal predicate significantly influences the runtime.
4) Still yet to research the dependence on number of actions and number of attributes.
5) Still yet to compare with the reference possible implementations of solvers using ENHSP and reinforcement learning.

Thank you for listening