Electronic Communications of the EASST
Volume 81 (2021)

9th International Symposium on Leveraging Applications of Formal Methods, Verification and Validation

-

Doctoral Symposium, 2021

Synthesising evolvable smart manufacturing scenarios

Ivan Guevara

14 pages

Guest Editors: Sven Jörges, Anna-Lena Lamprecht, Anila Mjeda, Stefan Naujokat
ECEASST Home Page: http://www.easst.org/eceasst/ ISSN 1863-2122

Synthesising evolvable smart manufacturing scenarios

Ivan Guevara

[1] ivan.guevara@ul.ie, http://www.ul.ie/
University of Limerick, Limerick, Ireland

Abstract: Smart manufacturing contexts have been experiencing an increasing complexity over the years, leveraged by higher computational power (CPU/GPU), increasing speed connections (5G/6G), AI transformation from use case to mass adoption and robotic systems enhancements. Research has been focusing on architectural issues to improve disposition of the components involved within a smart manufacturing scenario and find the most effective configuration to provide the most efficient production environment.
We introduce in this paper an improved version of our "maze generator", a navigation scenario generator that uses a machine learning approach from the Evolutionary Computing branch called Grammatical Evolution (GE) to automatically generate different scenarios with different configurations. GE takes advantage of a BNF grammar (rules describing the experiment) to define the search space, and genetic operations (crossover, mutation, selection, etc) to provide novelty to the solutions found. This not only enables the possibility to test autonomy, self-sufficiency and performance on a simplified model, but also to determine levels of difficulty to test the simulated navigation model under specific conditions.

Keywords: Smart Manufacturing, Machine Learning, Grammatical Evolution, Robotic Navigation

1 Introduction

Robotic navigation scenarios provide a way to implement, analyse, control, and test over a simplified model, to enable quick insights about the behaviour of our system. They are a key component of this research, being a cost-effective answer to the enquiries we might have during the process. These ones also allow us to tackle important issues, i.e, how the disposition of elements can be optimised, generalization capabilities of our controllers, incidence of selected raw materials to build real world scenarios, interaction between cobots, shape optimization for elements, etc. In this context, we also discuss which elements essentially constitute the "grammar of scenarios", how they can be associated to various concepts of complexity (based, e.g., on those spaces, cost of production of the scenario, skill levels needed to move around, and other perspectives), and how these different understandings can impact the experience of robots and humans navigating.

As mentioned before, one of the challenges in robotic navigation is to tackle the generalization problem, i.e., finding a robotic controller general enough to cover the different levels of complexity and navigate most of the presented scenarios the robot will have to face. Although significant work is found in that area, where we can mention [NUT17] and [NRG+20] show-

ing that controllers can be efficiently evolved themselves through GE using various strategies (controllers that prefer novelty search, moderated free exploration, were able to outperform the conventional objective-based search approach), we cannot apply the same criteria to evolvable scenarios. We still require a lot of the preparation and pre-training on prepared scenarios (the mazes of our title), to reach a level of navigation confidence sufficient to be ported to real world tasks. But what are the crucial characteristics of a scenario? What's the difficulty profile of a maze?

The paper is organized as follows: Section 2 covers related work in the application domain, maze generation, as well as our technology of choice, genetic algorithms, grammatical evolution and its relationship with robotic navigation. Section 3 will explain in the detail the experiment setup (maze grammar, hyperparameters, fitness function and software architecture) and results and lessons learned are reported in Sections 4 and 5 respectively.

2 Related Work

2.1 Scenario generation

Although lacks in variety with the type of figures they produce, we can find in the literature very efficient and broadly used algorithms that build 2D maze scenarios with high precision, such as Kruskal's Algorithm [Kru56], Prim's Algorithm [Pri57], Wilson's algorithm [Wil96] and Aldous-Broder algorithm [Ald90] [Bro89]. Other approaches for maze generation through evolution with more realistic output can be found in [SP10], where a FI2-2Pop genetic algorithm is used to evolve scenarios in popular video-games such as Mario Bros and Legend of Zelda, also [ALM11] shows the usage of Dynamic Programming for maze-like levels used in games, [TPY10] developed a multiobjective evolutionary algorithm to generate, not only strategy game maps, but to optimise experiences for different types of players, [Ora10] uses evolutionary algorithms in order to generate solutions for the Japanese game called "Shinro".

2.2 Grammatical Evolution

GE is a machine learning technique inspired in biology that uses mainly 3 stages (Fig. 1) in order to evolve solutions: 1) initializes a random population 2) using structural characteristics provided in Backus-Naur Form, BNF [BBG+63], maps those individuals (solutions) and generates a set of solutions in terms of the BNF Grammar (phenotypes) and then, evolves the population by utilising genetic operations (mutation, crossover, selection). 3) determines which are the most apt individuals to use them for the next generation. It works by successive approximation based on a population approach and tries to achieve better and improved individuals in each generation, improving the fitness score of the overall population. In this particular context, as shown in (Fig. 2), random population (characterized by integer arrays) goes through a mapping process with the BNF Grammar, where genotype is transformed into a combination of figures given by the grammar (phenotype). The process begins by grabbing the first element in the array, doing a MOD operation and the result of that operation corresponds to an index in the grammar previously defined. It will continue until the solution has derived all the elements (in this case, our final solution found was *square(eshape())*). It could be the case that a solution is not found,

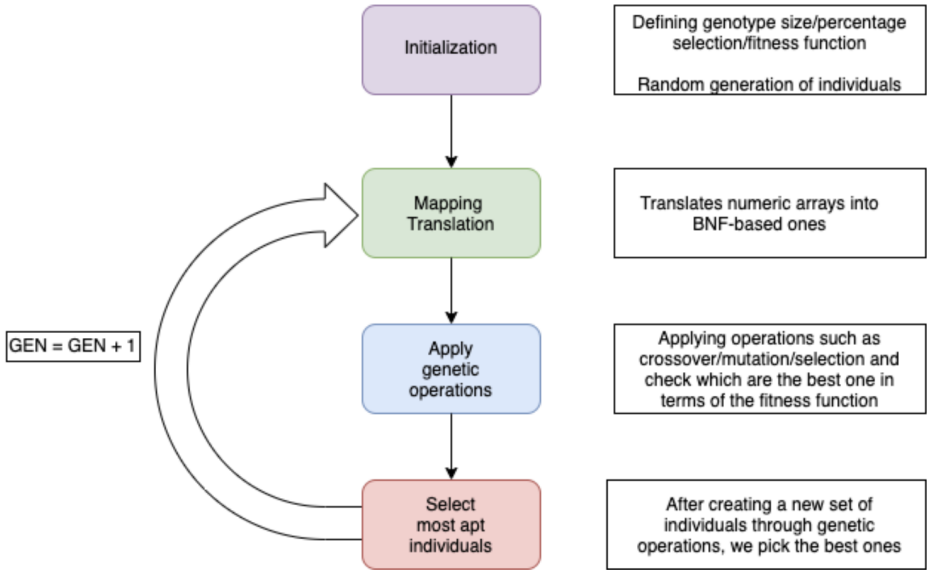

Figure 1: Generic workflow for a GE experiment

so we will mark our individual as invalid and discard it during the evolutionary process.

We can find several number of success cases where GE has been applied, such as communication networks [FLK+17] , search-based software engineering [CFR+17], program synthesis [OR99], sport analytics [FLK+17] and animation [Mur11]. It is one of the prominent success stories of Irish research in applied AI.

2.3 Robotic Navigation

GE has been used for robotic navigation, to mostly produce controllers that navigate predefined mazes. In [NRG+20] using NetLogo [Wil99] as a basis, controllers language were evolved for moving through the maze and reach the target, using two different strategies expressed by means of their respective fitness functions: The objective-based search (OS) rewarding those moves that get the agent closer to the target and novelty search (NS) rewards exploring new areas over proximity to the target. The general idea of the experiment was to see whether GE could enable a controller general enough to navigate the majority of the produced scenarios, and for that the controllers were trained with a dataset containing 60 maze instances and a test set with 100 different ones. The best controllers were chosen to be tested in a sort of cross-validation set with 15 more scenarios(Fig. 3) in order to reach the decision of which one was the best. As we can evidence in the same figure, we have increasing complexity in the maze configurations starting with easy, medium and hard. The easiest mazes are being represented by only 1 wall (brown large line) and the target point (red point), the ones with medium difficulty with 2 walls and the target point and finally the hardest ones with 3 walls and the target point. All the mazes were developed by hand, using the NetLogo scripting language, making 5 for each of them with code basically consisting of indicating coordinates, colours and disposition of each element, a quite

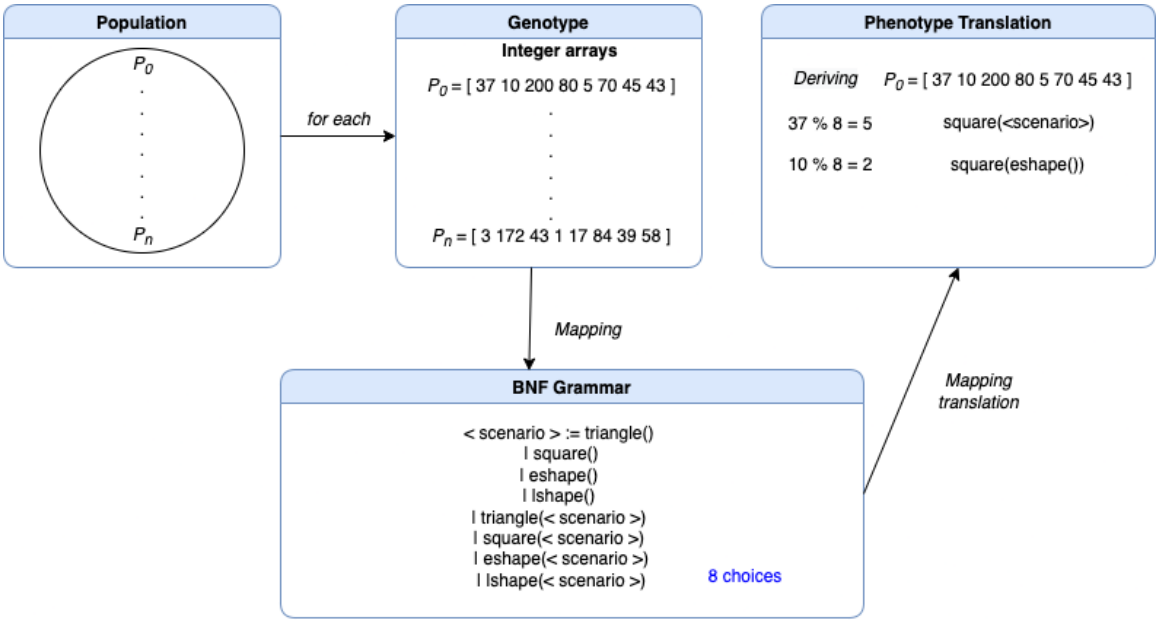

Figure 2: Step by step for the MazeGen experiment

complex task as the scripting language does not allow us to use high-level abstractions in an easy way. The development cycle was inefficient as we needed the exact coordinates for the elements, not having even the possibility to use high-order functions (Fig. 4)

2.4 Framework direction

MazeGen aims to support for scenarios: 1) automated generation, 2) increasing complexity generation, 3) an easy learning curve for rapid development, and 4) a library of available predefined building blocks ready to use. PyGEVO, the framework giving support to MazeGen, was conceived with a focus in a low-code fashion, trying to imitate different approaches available in the industry such as Ludwig [MDM19], Fastai [HG20], Nni [Mic21] or the Turicreate engine from Apple [App18]. MazeGen was developed aiming to bring a powerful tool that can easily bootstrap an experiment, trying to put the application expert (a potential non-programmer) in the center of the development process, instead of relying on the manual reuse of boilerplate code, which is hard to maintain and scale. We refer here as a blueprint to the jABC framework [SMN+06], a development framework which follows the XMDD (eXtreme Model-Driven Design) [MS12] and the OTA (One Thing Approach) [MS09]. It has 2 development levels: a modelling level where models are defined in a high-level abstraction layer, and the implementation level, where the actual program artifacts are created. The idea behind the MazeGen framework is to follow the same organization structure as jABC: we do not only interact with high-level abstractions (models), but the whole workflow experiment can be accomplished by using a few lines of code and easily extended. This is appropriate for people with no experience in programming languages,

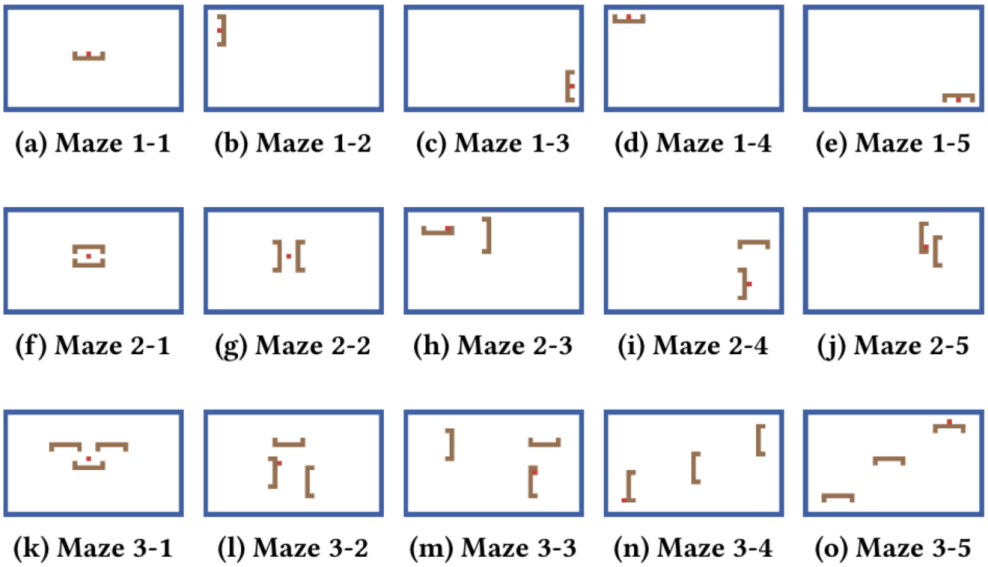

Figure 3: Mazes generated for validation process

as shown in [LT14] and [Tat20] in various learning and adoption contexts.

3 Experimental Setup

In order to concretely define the experiment, we need to work on 4 main issues: 1) Define the rules for the BNF grammar to limit the universe of solutions we want to have. 2) Developing a software architecture capable of dealing with the graphical framework and, at the same time, providing a friendly and easy way to interact with the API. 3) Hyperparameters, defining the number of runs/population size/genetic operations to apply to the ongoing experiment 4) Fitness function, which will define the criteria to look for the most apt individuals.

3.1 BNF grammar

Representing the composability of elements for creating the scenario, in the context of our BNF grammar, we might see different figures that will take part of it. The majority of the figures are a geometric element composed by dots $(x_1,y_1),(x_2,y_2),.....,(x_n,y_n)$ where $x_1,x_2,.....,x_n$ and $,y_1,y_2,.....,y_n \in Z$ and $n \in N$. The geometric figures are the following ones:

- <scenario>: This is the first and unique rule from the BNF grammar. It will contain all the figures allowed in the scenario and their respective recursive forms.
- Triangle: A six dot element placed in random coordinates with a random size.
- Square: An eight dot element also placed with random coordinates and a random size, representing a simple obstacle to the robot.

```
if pxcor = 37 and pycor >= 14 and pycor <= 15
  [ set pcolor red ]                              ;; ... draws left edge in red
if pxcor = 28 and pycor >= 14 and pycor <= 15
  [ set pcolor red ]                              ;; ... draws right edge in red
if pycor = 15 and pxcor >= 28 and pxcor <= 37
  [ set pcolor red ]                              ;; ... draws upper edge in red
;; We create the bottom wall
if pxcor = 28 and pycor >= 10 and pycor <= 11
  [ set pcolor red ]                              ;; ... draws left edge in red
if pxcor = 19 and pycor >= 10 and pycor <= 11
  [ set pcolor red ]                              ;; ... draws right edge in red
if pycor = 11 and pxcor >= 19 and pxcor <= 28
  [ set pcolor red ]                              ;; ... draws upper edge in red
;; We create the bottom wall
if pxcor = 24 and pycor >= 5 and pycor <= 6
  [ set pcolor red ]                              ;; ... draws left edge in red
```

Figure 4: A fragment of the Netlogo scripting code used for Maze development

- LShape: Also an eight dot element placed in random coordinates with also random size, representing a simple trap to avoid the pass.
- EShape: A sixteen dot element placed in random coordinates with also random size, representing a conjunction of walls working as a trap to avoid the pass of the robot, a more complex trap that the one before.
- Recursive forms: a composable element where any kind of combination between the ones mentioned before is allowed, i.e., triangle(square), triangle(square(lshape)), etc.

3.2 Fitness Function

Two basic criteria were used for the fitness function in order to pick the most performing individuals through the evolutionary process during the experiment:

1. more geometric elements $\rightarrow$ better individual
2. more geometric elements + size filters $\rightarrow$ better individual

The first criterium was too simplistic as we only counted the number of elements the individual could reach and select those ones for the next generation. Then we applied genetic operations (crossover, mutation, selection) and continued with the evolutionary process. That fitness function created some issues as there was a tendency to create big objects, as big as the whole scenario, so we decided to modify it and have a second version. The improved function took into account bigger objects and penalized all the individuals whose elements were above 500 pixels. This is still a simplistic approach, as we don't take into account solvability of the maze, real world scenarios and targeted complexity , but it was a good start as a preliminary approach.

3.3 Hyperparameters

Each experiment was run 10 times to have enough information to reach reliable conclusions for each configuration. In terms of hyperparameters, we carried out 6 sets of experiments with

different configurations reported in Table 1. All the parameters were defined based on a range of numbers, this way we could appreciate how the experiment evolved :

1. Individuals: Represented by an array of numbers (binary or integer), these ones will be part of the solution once they go through the mapping process.
2. Genotype size: Size for the array of numbers. Representing a ”genetic representation” of the individual.
3. Generations: The amount of time within the experiment.
4. P. Selection: A percentage of selection defining the number of individuals that will go to the next generation.
5. F. Function: Defines the criteria why an individual is better than other.

Table 1: Experiments set up: hyperparameters and option configuration for the 6 sets of experiments.

Hyperparameters	**Experiment 1**	**Experiment 2**	**Experiment 3**	**Experiment 4**	**Experiment 5**	**Experiment 6**
Runs	10	10	10	10	10	10
Individuals	1000	5000	1000	5000	10000	50000
Genotype Size	32	32	32	32	32	32
Generations	15	15	15	15	30	30
P. Selection	0.1	0.1	0.1	0.1	0.1	0.1
F. Function	Higher→ better	Higher → better	Penalize	Penalize	Penalize	Penalize

3.4 Software Architecture

Having a highly simplified grammar with one rule for the scenario composition is not enough, we need a ”bridge” between our machine learning models and the graphical representation. To accomplish that, in the first version of this paper, we defined a software architecture where external components took care of the attribute evaluation, fitness evaluation and serialization of graphical components. Despite the fact it was a good approach, we lacked of control for the different elements drawn in the scenario as we didn't track elements, we just rendered points. Having control of those ones is key to have a more easy way to interact with them and allow a more straightforward handling of new behaviours such as collision detection, color definition, rotation, etc. In particular, we created several data classes in order to contain the graphical representations: Layout, Eshape, Lshape, RectangleShape, TriangleShape, ScenarioBuilder. Each of those objects will represent an element, giving each of them the responsibility for their own actions. As shown in 5, the experimental set up still uses PyGEVO [IL], a state-of-the-art Grammatical Evolution framework, to manage the BNF and the genotype to phenotype evaluation, and Kivy [MGA+20], a free and open-source Python-based graphical framework that we use to visualize the scenarios. Both are chosen because they are open source and provide tools for rapid development, which led us to a rapid bootstrapping of the development process. We can also see the interaction between the ScenarioBuilder and our data classes, being the one in charged of generating those instances.

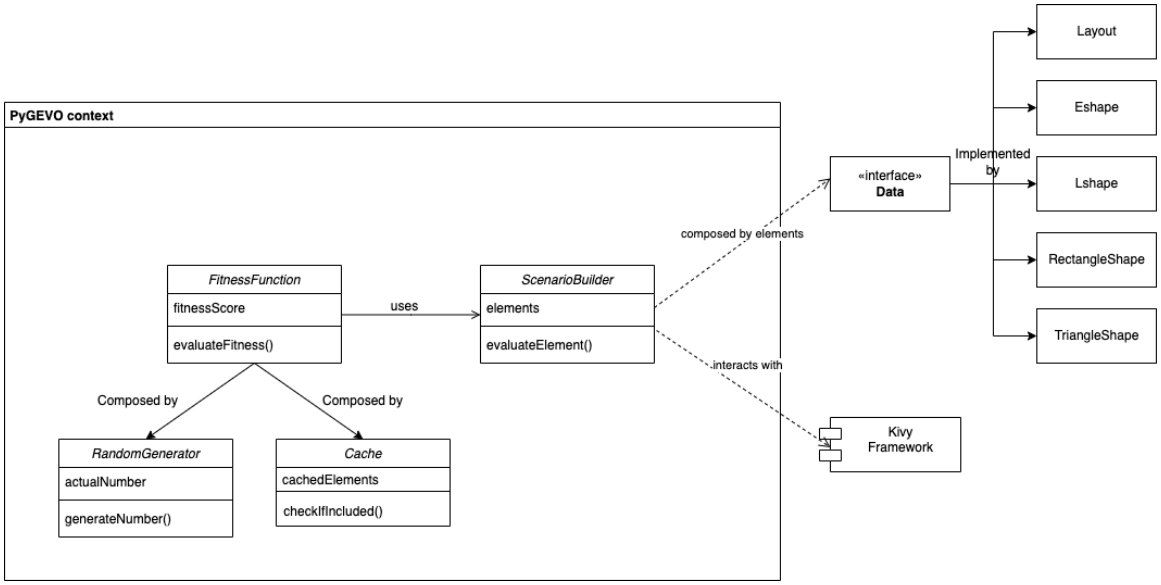

Figure 5: The new architecture developed for the 2nd version of the MazeGen (UML notation)

The main orchestration occurs in a python script which defines the fitness function and invokes PyGEVO. PyGEVO abstracts all the logic for the evolutionary process and the synthesis of the phenotypes, handling the grammar-related part of the GE approach. The cache also helps us to achieve a better performance when dealing with individuals. As soon as our phenotypes are derived, they are translated into Kivy code and placed into a proper log. For this, we created a simple Domain Specific Language for the Scenario Builder's interface, to easily handle the creational part of the elements.

4 Results

The results improved with the new version of the architecture as we were able to manage more easily the creational part of each element. Experiments in general had between 14 and 20 geometric figures with diverse complexity, this is due to the relationship between the simplicity of the grammar and the genome size of 32 codons: it takes a low number of codons to completely describe an object with this grammar, therefore it tends to derive phenotypes with a large number of elements. In turn, the length of the genome relates to the computational time, so we see that there is a trade-off between the expressiveness of a grammar in terms of number of productions and the computational effort needed to derive scenarios of a sufficient or good complexity taking [NRG+20] as a reference for scenarios of sufficient and good quality.

In Fig. 6 and Fig. 7 we obtained 2 scenarios with 16 and 17 elements respectively, most of them being triangles located in random locations. We can also observe black squares and a rectangle, acting as a big obstacle. Fig. 8 and Fig. 9 show a 18 and 19 element scenario generated: we see several elements distributed across the scenario acting as a container wall, composed by triangles,

squares and lshapes. For each of the figures located in the these scenarios, we have a correlated object describing a bounding box. This would make possible, despite the complexity observed, to use them to train robotic controllers. We still face an issue with the fitness function: despite its improvement by considering the large sized obstacles, we still do not take into account important details such as resolution of the maze, real world components or equivalence classes to address complexity. This has a clear impact on the quality of the solutions that we could provide to a real-world scenario, but we can still reach a good level of complexity with the building blocks defined in the grammar.

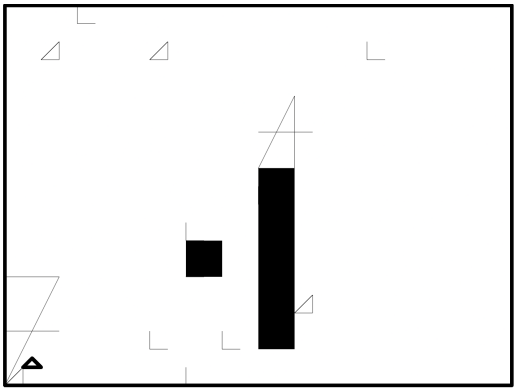

Figure 6: Scenario 1

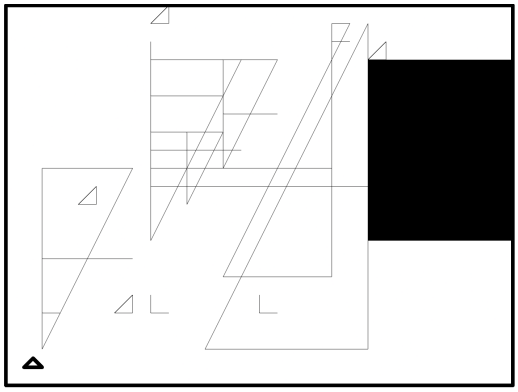

Figure 7: Scenario 2

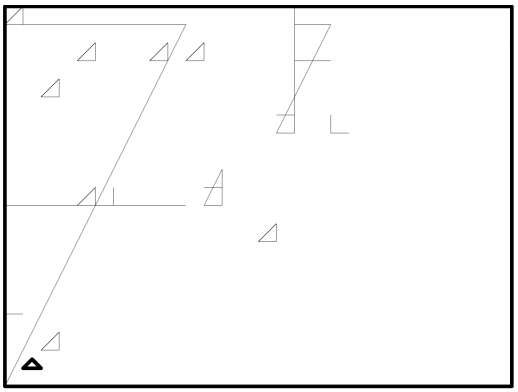

Figure 8: Scenario 3

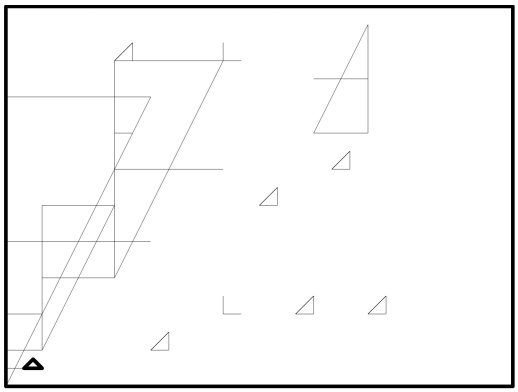

Figure 9: Scenario 4

5 Future Work

The preliminary work accomplished with MazeGen evidences a novel way to recreate maze scenarios with an evolutionary machine learning approach, leveraging the rules and constraints described in a BNF grammar and the corresponding fitness function. The approach itself, despite the good results obtained in the first version of the framework, still needs improvement as it shows some problems concerning the solvability of the scenario and the lack of connection of the figures with a real world case study. In this context, a new version of the MazeGen (v2.0) is

being developed with a new grammar representing new, more realistic elements tied to a specific application domain (e.g., in a manufacturing floor, the stations of a production line, a cobot, a router, a vertical and horizontal wall) and a fitness function capable of penalizing overlapping objects, in order to have a better distribution between the elements of the robotic scenario. In this new version, the elements (still represented using the Kivy framework) are placed around the scenario and the fitness score will diminish in relation to the amount of overlapped figures it contains. As we can see in Fig. 10 and Fig. 11, the resulting scenarios have a better disposition, with an improved "semantic meaning": On the left, we see 2 vertical walls (black) and 3 production lines (blue and yellow background), and the right one is composed by an horizontal wall, 3 production lines and a router (represented by a yellow square).

This work is in line with the different tools we have been developing as a research group, for instance the self-contained model-driven solution in the context of a Digital Thread Platform for CPS [MCG+21], where the general idea of this platform is to enable a high-level abstraction layer sufficient enough to orchestrate different levels of heterogeneity, being capable of easily plug solutions found in the industry and include them in the development cycle as self-contained models. We can also evidence different projects such as [CGJ+22], [CM21] or [JGMP21] concerning various other smart manufacturing aspects, where the main focus is put on model-driven architectures that can be easily bootstrapped in a simple fashion way, not having to deal with boilerplate code, but with models. The logic behind these tools goes through a model-to-code transformation of the workflows, where single functionalities are placed in a SIB (Service-Independent Building block), that can be used in an environment like DIME [BFK+16], a friendly model-oriented intuitive IDE.

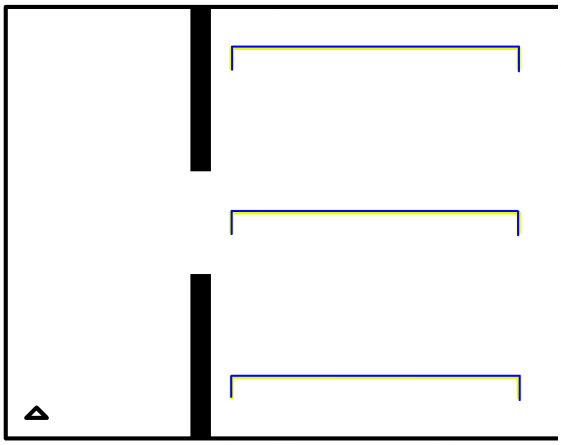

Figure 10: Scenario 1 - version 2

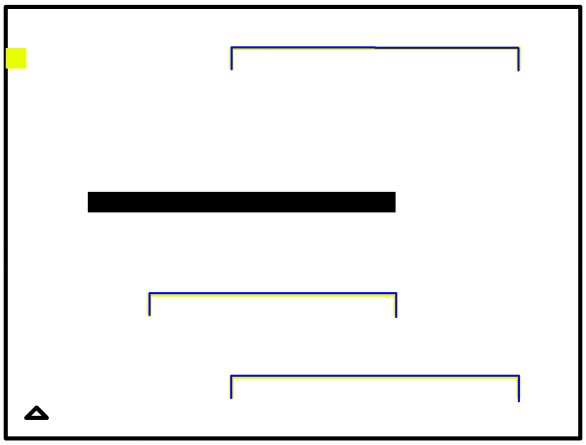

Figure 11: Scenario 2 - version 2

6 Lessons Learned and Conclusions

Evolving robotic navigation scenarios through GE is a novel way to have systematic and automatic scenario development, providing a variety in terms of elements that compose each maze,

giving control on the global settings and tracking the different elements with their corresponding coordinates. A new architecture was built to accomplish the aforementioned findings, but it took quite a time to re-adapt the building of each scenario in order to have, not only coordinates drawn in the scenarios, but objects to interact with.

Despite the known issues from the fitness function (e.g., possibility to specify some predefined complexity, correlate complexity and configuration inside a scenario, as well as the overall solvability itself), we managed to give a great step towards it, as we replace ordinary coordinates in a x-y plane for objects that can address behaviour in order to have more complex interactions among them. Enabling control over the figures will allow detecting collisions, which will deliver the possibility to filter also those who cannot be solved, improving the accuracy of the overall experiment.

Acknowledgements: This work was supported by the Science Foundation Ireland grant 16/RC/3918 (Confirm, the Smart Manufacturing Research Centre).

Bibliography

[Ald90] D. J. Aldous. The Random Walk Construction of Uniform Spanning Trees and Uniform Labelled Trees. *SIAM Journal on Discrete Mathematics* 3(4):450–465, 1990.
doi:10.1137/0403039
https://doi.org/10.1137/0403039

[ALM11] D. Ashlock, C. Lee, C. McGuinness. Search-Based Procedural Generation of Maze-Like Levels. *IEEE Transactions on Computational Intelligence and AI in Games* 3(3):260–273, 2011.
doi:10.1109/TCIAIG.2011.2138707

[App18] Apple. Turicreate Engine. 1 2018.
https://github.com/apple/turicreate

[BBG+63] J. W. Backus, F. L. Bauer, J. Green, C. Katz, J. McCarthy, P. Naur, A. J. Perlis, H. Rutishauser, K. Samelson, B. Vauquois, J. H. Wegstein, A. van Wijngaarden, M. Woodger. Revised report on the algorithmic language ALGOL 60. *The Computer Journal* 5(4):349–367, 01 1963.
doi:10.1093/comjnl/5.4.349
https://doi.org/10.1093/comjnl/5.4.349

[BFK+16] S. Boßelmann, M. Frohme, D. Kopetzki, M. Lybecait, S. Naujokat, J. Neubauer, D. Wirkner, P. Zweihoff, B. Steffen. DIME: a programming-less modeling environment for web applications. In *International Symposium on Leveraging Applications of Formal Methods*. Pp. 809–832. 2016.

[Bro89] A. Broder. Generating random spanning trees. In *30th Annual Symposium on Foundations of Computer Science*. Pp. 442–447. 1989.
doi:10.1109/SFCS.1989.63516

[CFR+17] B. Cody-Kenny, M. Fenton, A. Ronayne, E. Considine, T. McGuire, M. O'Neill. A Search for Improved Performance in Regular Expressions. In *Proceedings of the Genetic and Evolutionary Computation Conference*. GECCO '17, p. 1280–1287. Association for Computing Machinery, New York, NY, USA, 2017.
doi:10.1145/3071178.3071196
https://doi.org/10.1145/3071178.3071196

[CGJ+22] H. A. A. Chaudhary, I. Guevara, J. John, A. Singh, T. Margaria, D. Pesch. Low-code Internet of Things Application Development for Edge Analytics. In Camarinha-Matos et al. (eds.), *IFIP Advances in Information and Communication Technology*. Springer International Publishing, 2022.

[CM21] H. A. A. Chaudhary, T. Margaria. Integrating external services in DIME. In *International Symposium on Leveraging Applications of Formal Methods*. Pp. 41–54. 2021.

[FLK+17] M. Fenton, D. Lynch, S. Kucera, H. Claussen, M. O'Neill. Multilayer Optimization of Heterogeneous Networks Using Grammatical Genetic Programming. *IEEE Transactions on Cybernetics* 47(9):2938–2950, 2017.
doi:10.1109/TCYB.2017.2688280

[HG20] J. Howard, S. Gugger. Fastai: a layered API for deep learning. *Information* 11(2):108, 2020.

[IL] G. Ivan, G. Lucas. PyGEVO.
https://github.com/IvanHGuevara/PyGEVO

[JGMP21] J. John, A. Ghosal, T. Margaria, D. Pesch. DSLs for Model Driven Development of Secure Interoperable Automation Systems with EdgeX Foundry. In *2021 Forum on specification Design Languages (FDL)*. Pp. 1–8. 2021.
doi:10.1109/FDL53530.2021.9568378

[Kru56] J. B. Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society* 7(1):48–50, 1956.

[LT14] A.-L. Lamprecht, M. Tiziana. *Process design for natural scientists*. Springer, 2014.

[MCG+21] T. Margaria, H. A. A. Chaudhary, I. Guevara, S. Ryan, A. Schieweck. The Interoperability Challenge: Building a Model-Driven Digital Thread Platform for CPS. In Margaria and Steffen (eds.), *Leveraging Applications of Formal Methods, Verification and Validation*. Pp. 393–413. Springer International Publishing, Cham, 2021.

[MDM19] P. Molino, Y. Dudin, S. S. Miryala. Ludwig: a type-based declarative deep learning toolbox. 2019.

[MGA+20] V. Mathieu, P. Gabriel, A. Akshay, E. Matthew, T. Alexander, L. Richard, B. Peter, A. Sebastian, S. Terje, C. Ilya, T. Kathryn, H. Thomas, D. Christopher, K. Jacob, R. Ben, R. Philip, U. Meet, Mitō), K. Rafał, B. Guillaume, M. Andre, P. Jeff, P. T.

Karl, M. Charles, F. Ian, M. Edwin, K. Brian, P. Dusty, S. Zachary, L. S. Kristian, J. Paige, G. Mirko, O. M. Edwin, X. X. Jason, H. Ben, B. Sam, N. Mihai, S. Azim, B. Docimique, G. Mikhail, D. Denys, W. Kjell, B. Julien, S. Ethan, L. Dominik, C. Vernon, N. Robert, K. Joseph, A. Ismael, S. Eric, J. Abhinav, M. Stuart, P. Stéphane, M. Jim, E. Roger, J. Richard, O. Oleksandr, W. Björn, L. Alex, R. Victor, L. Sander, H. Rene, S. Niko, K. Don, C. David, G. Cayci, J. Alan, M. Mihály, M. Oliver, O.-E. Ng, S. Karl, M. Julien, B. Jakub, V. Gabriel, R. Clément, K. Albert, O. Shayne, E. Godwin, A. M. Ronnie, P. Rasmus, W. Phunsuk. Kivy. Dec. 2020.
doi:10.5281/zenodo.5097751
https://doi.org/10.5281/zenodo.5097751

[Mic21] Microsoft. Neural Network Intelligence. 1 2021.
https://github.com/microsoft/nni

[MS09] T. Margaria, B. Steffen. Business process modeling in the jABC: the one-thing approach. In *Handbook of research on business process modeling*. Pp. 1–26. IGI Global, 2009.

[MS12] T. Margaria, B. Steffen. Service-orientation: conquering complexity with XMDD. In *Conquering complexity*. Pp. 217–236. Springer, 2012.

[Mur11] J. E. Murphy. *Applications of evolutionary computation to quadrupedal animal animation*. PhD thesis, University College Dublin, 2011.

[NRG+20] E. Naredo, C. Ryan, I. Guevara, T. Margaria, P. Urbano, L. Trujillo. *General Controllers Evolved through Grammatical Evolution with a Divergent Search*. P. 243–244. Association for Computing Machinery, New York, NY, USA, 2020.
https://doi.org/10.1145/3377929.3390059

[NUT17] E. Naredo, P. Urbano, L. Trujillo. The training set and generalization in grammatical evolution for autonomous agent navigation. *Soft Computing* 21(15):4399–4416, Aug 2017.
doi:10.1007/s00500-016-2072-7
https://doi.org/10.1007/s00500-016-2072-7

[OBF16] M. O'Neill, A. Brabazon, D. Fagan. An exploration of grammatical encodings to model six nations rugby match outcomes. In *2016 IEEE Congress on Evolutionary Computation (CEC)*. Pp. 4429–4436. 2016.
doi:10.1109/CEC.2016.7744353

[OR99] M. O'Neill, C. Ryan. Automatic generation of caching algorithms. *Evolutionary Algorithms in Engineering and Computer Science* 30:127–134, 1999.

[Ora10] D. Oranchak. Evolutionary algorithm for generation of entertaining shinro logic puzzles. In *European Conference on the Applications of Evolutionary Computation*. Pp. 181–190. 2010.

[Pri57] R. C. Prim. Shortest connection networks and some generalizations. *The Bell System Technical Journal* 36(6):1389–1401, 1957.

[SMN+06] B. Steffen, T. Margaria, R. Nagel, S. Jörges, C. Kubczak. Model-driven development with the jABC. In *Haifa verification conference*. Pp. 92–108. 2006.

[SP10] N. Sorenson, P. Pasquier. Towards a Generic Framework for Automated Video Game Level Creation. In Di Chio et al. (eds.), *Applications of Evolutionary Computation*. Pp. 131–140. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[Tat20] A. Tatnall. *Encyclopedia of Education and Information Technologies*. Springer, 2020.

[TPY10] J. Togelius, M. Preuss, G. N. Yannakakis. Towards Multiobjective Procedural Map Generation. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*. PCGames '10. Association for Computing Machinery, New York, NY, USA, 2010.
doi:10.1145/1814256.1814259
https://doi.org/10.1145/1814256.1814259

[Wil96] D. B. Wilson. Generating Random Spanning Trees More Quickly than the Cover Time. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*. STOC '96, p. 296–303. Association for Computing Machinery, New York, NY, USA, 1996.
doi:10.1145/237814.237880
https://doi.org/10.1145/237814.237880

[Wil99] U. Wilensky. NetLogo (Version 4.1. 2)[Software]. *Center for Connected Learning and Computer-Based Modeling, Northwestern University. Illinois: Evanston. Available from http://ccl. northwestern. edu/netlogo/APPENDICES*, 1999.