International Journal on Artificial Intelligence Tools

A Short Introduction to Procedural Content Generation Algorithms for Videogames

Nicolas A. Barriga
School of Videogames Development and Virtual Reality Engineering
Universidad de Talca
2 Norte 685, Talca, Región del Maule, Chile
nbarriga@utalca.cl

Received (Day Month Year)
Revised (Day Month Year)
Accepted (Day Month Year)

One of the main costs of developing a videogame is content creation. Procedural Content Generation (PCG) can help alleviate that cost by algorithmically generating some of the content a human would normally produce. We first describe and classify the different types of content that can be automatically generated for a videogame. Then, we review the most prominent PCG algorithms, focusing on current research on search-based and machine learning based methods. Finally, we close with our take on the most important open problems and the potential impact solving them will have on the videogame industry.

Keywords: Procedural Content Generation; PCG; Videogames; Computer Games; Game Design.

1. Introduction

Procedural Content Generation (PCG) is the automation of media production. This media can be anything a human would usually author, such as poetry, paintings, music, architectural drawings or film. PCG for games is the use of algorithms to produce game content that a designer would typically create, such as textures, sound effects, maps, levels, characters, weapons, quests or even game mechanics and rules. We will focus on content that directly relates to game mechanics and player interaction, that is, on *functional*, rather than *cosmetic* content. In particular, we are interested in what Hendrikx *et al.*[1] calls game space, systems and scenarios, which includes maps, ecosystems, levels and stories (among others). This doesn't mean that cosmetic content isn't important—what would the *Super Mario Bros.* franchise be without it's iconic character?—but rather a reflection on what the current research needs are. Procedural generation of cosmetic content has received plenty of attention from different communities (mainly from graphics, visualization and sound design) over the last five decades, whereas functional content generation is in its infancy.

One of the main costs of developing a videogame is content creation, with some researchers[1] estimating it to be around 30%–40% of the US$20M–US$150M average budget for AAA games. This content production requires highly specialized personnel, limiting the possibilities of small studios and increasing costs in bigger companies. Fortunately, PCG techniques enable the algorithmic creation of content without (or with limited) human effort.

This area of Artificial Intelligence (AI) saw its first applications in videogames in the late 70s and early 80s, in games such as *Rogue* and *Elite*. In the first one, the goal was replayability—generating novel content every time the game is started—, while in the latter, the goal was compression—it takes less data to save the seeds for a pseudo-random number generator than the sequence of numbers generated. Almost 40 years after these first examples, PCG applications remain confined to small niches (textures, dungeons) and are so uncommon, that the sole mention of PCG being used in a project can turn them into worldwide (specialized) media sensations, like *No Man's Sky*, with it's virtually infinite generated universe. This lack of widespread adoption has three main causes: inadequate algorithms, lack of tools, shortage of specialized professionals.

The algorithms for more basic content, such as textures and vegetation are very mature, and as such, are widely used. However, as one goes up the content hierarchy, towards generating levels, dialogue, story, mechanics or rules, current methods are more primitive. There are some experimental systems available, but little commercial adoption. Similarly, off-the-shelf, ready to use tools are only available for basic content. This is partly because the algorithms are not ready, but also due to the difficulty in adapting general purpose algorithms to work seamlessly in very different projects. Finally, with PCG in general—and PCG for videogames in particular—being a relatively new and niche field, most formal education for computer scientists and software engineers doesn't include it. Which leads to the industry relying on the few tools available, and no cross-pollination between academic research and industrial development.

2. PCG Classification

Game content that can be procedurally generated has been classified[1] in six types:

(1) **Game Bits** are the fundamental units of game content. They can be *necessary*, such as the texture differentiate the main character from the enemies, or *optional*, such as background music, but either way, they usually don't interact with the player directly when considered independently. Game bits include textures, sound, vegetation, architecture and graphics effects (fire, water, etc.).
(2) **Game Space** is the environment in which the game takes place, and includes maps and terrain.
(3) **Game Systems** simulate more complex environments such as ecosystems, road networks, urban environments and entity interactions.

(4) **Game Scenarios** organize the previous levels into coherent plans or sequence of events. They include puzzles, stories and levels.
(5) **Game Design** is the set of rules and mechanics of the game.
(6) **Derived Content** can be created as a companion to the game world, like leaderboards and news items covering player actions in the game world.

Togelius *et al.*[2] presents, and Shaker *et al.*[3] later expands, a taxonomy of content generation. As previously mentioned, some content is *necessary*, while other is *optional*. It can be generated *online* during execution of the game, or *offline*, during development. Some algorithms just take a random seed which completely determines their generated content, while others offer a greater *degree of control*. The generated content can be *stochastic*, changing with every run of the algorithm, or *deterministic*, allowing for the re-generation of the same content multiple times. The generation process might be *constructive*, with a built-in guarantee on the quality of the content, or it might use a *generate-and-test* approach, that keeps running until the content satisfies some predefined property. Generation can be *generic*, or *adaptive*, creating content specially tailored to each player. Finally, the process can be completely *automatic*, or human interaction may be allowed (or required) in a *mixed-initiative* generator.

Several classification schemes have been proposed to group PCG algorithms[3,1,4,5] with usually four to six categories. For the purposes of this survey, we are going to group them in three categories: traditional methods, search-based methods and machine learning methods. Traditional methods include pseudo-random number generators and constructive methods, generative grammars, fractals and noise, cellular automata and others. Search-based methods include artificial intelligence and operations research areas, such as heuristic search, local search and optimization. Machine learning includes supervised learning, unsupervised learning and reinforcement learning. Keep in mind that both the classification and the examples in each category are somewhat arbitrary, but serve to further narrow our focus.

Some of the most desired and elusive qualities of a good PCG algorithm are controllability, expressivity and believability. We need methods that provide human designers some control over the generated content, that can express a wide range of content without seeming repetitive but at the same time without feeling haphazard, and that generate believable content.

Current AI-based generators have been shown to surpass their traditional counterparts in these aspects[6,7], though they *may* lag behind in speed and reliability, usually not providing any hard guarantees about the generated content.

2.1. *Traditional Methods*

Pseudo-random number generators and other constructive methods where the first to be applied in commercial videogames, and they have been widely used in dungeon and labyrinth generation[6]. Its main advantages are simplicity and speed. They are usually deterministic, generating the same content given the same seed. For this

reason, they were first used as a method of data compression, were every piece of content (usually a level) could just be represented as a single number: the seed to generate it.

Generation by fractals and noise is mainly used to produce content that must simulate the results of natural processes, such as mountains created by geological processes, or rock or vegetation textures. Content generated by this class of algorithms produce an *organic* feeling, and as such, they have been extensively used in games featuring landscapes and to generate textures for non-game applications.

The use of grammars to generate vegetation is probably the most successful case of videogame PCG. There are numerous established tools (such as SpeedTree[a], and their use in games, both AAA and independent, is widespread. This is an example of techniques who have migrated from videogames to other areas, like film or architecture. Other areas that could benefit commercially from grammar-based PCG, are city generation[8] and platformer games level generation[7,9].

2.2. *Search-based Methods*

Most of the PCG academic research focus in the past decade has been on search-based methods[2]. These techniques fall within the scope of generate-and-test methods, i.e., they generate content, and then evaluate it. However, rather than accepting or rejecting it, they score it. An evolutionary algorithm or other search/optimization procedure will keep producing content until a separate algorithm assesses that it meets some predefined criteria, or it's score is high enough.

Search-based methods are usually defined by three components: the space representation, the evaluation function, and the search algorithm. The representation has to balance size, reachability and locality. A very direct encoding will be able to represent all of the possible solutions, while maintaining good locality—a small change in the representation space translates to a small change in the solution space. However, for most interesting problems, the size will be unmanageable (a 100x100 tiles maze encoded in a vector of length 10,000 is larger than most search algorithms can handle). On the other end, a highly indirect encoding, like a list of desired properties of the solution, or even just a random number seed, produce a very small search space, but with very little locality and will leave large parts of the solution space unexplored. A balance has to be struck, but where exactly that balance lies, is deeply dependent on the problem and the algorithm used to solve it.

Once a solution has been generated, it has to be evaluated to assess its suitability. Evaluation functions can be direct functions, like a simple linear combination of features or a general function approximator (such as neural networks) trained on real data. They can also involve simulations of the game, where an artificial agent dynamically interacts with the generated content. Still, they can be interactive,

[a] *https://store.speedtree.com/*

involving a human player, and either explicitly asking her for an evaluation, or implicitly collecting data on the interactions.

Finally, a search algorithm has to explore the representation space and find the highest evaluated candidate solutions. The vast majority of search-based content generation research uses evolutionary algorithms. Table 1 shows a non-exhaustive summary of the most notable techniques found in the literature. Game design, at the top of the content hierarchy, stands out for only using direct representations and simulation-based evaluation functions, when intuition would tell us that the more complex content would lead to vast search spaces and slow simulations. However, this is misleading, as only rules for very simple games have been generated. If we take those out, current research seems to favour indirect representations and direct evaluation functions for the most complex content remaining (game levels).

Table 1. Search-based game content generation

Content Type	**Generated Content**	**Repr.**	**Evaluation**	**Ref.**
Game Bits	Weapons in *Galactic Arms Race*	Indirect	Implicit interactive	[10,11]
	Buildings	Indirect	Explicit interactive	[12]
Game Space	Racing game tracks	Direct	Simulation, NN	[13,14]
	Terrain	Indirect	Direct	[15]
	RTS maps	Direct and indirect	Direct and simulation	[16,17]
Game Systems and Scenarios	Number and board puzzles	Direct	Search-based simulation	[18,19]
	Super Mario Bros. levels	Indirect	Direct, NN	[20]
	Generic game levels	Indirect	Direct	[21]
Game Design	Board game rules	Direct	Simulation	[22]
	Grid game rules	Direct	Simulation	[23]
	Board game rules	Direct	Simulation and direct	[24]

2.3. *Machine Learning*

Most supervised machine learning (ML) problems are classification or prediction problems. That is, after learning from a labelled dataset, the goal is to be able to correctly classify (or predict a dependent variable of) a previously unseen input. Still, there are several successful ways of using ML techniques for generation of new content: Recurrent neural networks[25], Autoencoders[26], Generative Adversarial Networks (GANs)[27] and Markov Models[28] being the most famous ones.

Neural networks have seen a recent resurgence in interest,punctuated by the huge success achieved using deep Neural Networks for image classification[29] in 2012. Since then, they have helped reach new milestones in fields as diverse as speech

recognition[30] and board game playing[31,32]. PCG is no different, as can be seen in table 2. As almost all generated content in this table corresponds to game levels, we are going to further subdivide it in sequences, grids and graphs. Most 2D platformer games are traversed linearly from left to right, leading to a natural sequential representation of the level as a sequence of vertical slices. Games such as card games can also be represented as sequences. Game levels in Real-Time Strategy (RTS), Role-Playing Games (RPG) and open-world games can usually be represented as grids. More complex game levels can be represented by more arbitrary graphs.

Table 2. ML-based game content generation

Level Type	**Game**	**Algorithm**	**Training Data**	**Ref.**
Sequence	Super Mario Bros.	LSTM	Original game levels; simulated and human player trayectories.	[25,33]
	Super Mario Bros., Loderunner, Kid Icarus.	Multi-dimensional Markov Chains (MdMC), Hierarchical MdMC	Original game levels	[34,35] [36]
Grid	Super Mario Bros., Loderunner, Kid Icarus.	Markov Random Field	Original game levels	[28]
	Super Mario Bros.	Autoencoders	Original game levels	[26]
	StarCraft II resource locations	CNN	Human authored maps	[37]
Graph	Super Mario Bros.	K-means clustering	Gameplay videos	[38]
	Generic interactive fiction	OPTICS clustering	Crowdsourced stories	[39]

3. Open Problems and Outlook

Previous research has identified several open problems in the field[1,2,5]. We believe the most interesting ones to commercial videogame development are:

- Generate content at the top of the hierarchy. PCG at the systems and scenarios layers haven't seen much commercial use, except for early and simple games (rogue-likes), or very notable exceptions (No Man's Sky). The top of the hierarchy (Scenarios and up) has been mostly explored academically and in very constrained and simple games.
- More detailed generation. In several areas of game bits and space generation, PCG is the norm (trees, terrain), while in some others, human authored content still rules, even though PCG systems exist (maps, cities). The main different is the level of detail that those systems can generate. SpeedTree can generate

more detailed forests than any dedicated team of designers ever could, but the best city generation algorithms still generate extruded building profiles, with generic looking textures.

- Guaranteeing playability. A major impediment to widespread adoption of game systems, scenario or design procedural generation is the lack of guarantees that most search-based or machine learning approaches offer. This is not an issue for offline or mixed-initiative generation, but it is a show-stopper for online generation. Carefully constructed indirect representations could ensure playability.
- Personalized content. A task no human can perform, but an online algorithm could, is personalized content generation. By learning an internal player model, a game could generate content specially tailored to each user, implicitly evaluating the new content during gameplay and adjusting accordingly. Such as system has been used successfully to generate weapons in *Galactic Arms Race* and to tailor attacking waves in *Left for Dead*, but not to full game levels.
- Data scarcity. Supervised machine learning techniques perform best when there is abundant data available to learn from. If we wanted to generate extra levels for a particular game, only those few levels that come with the game could be readily used for training. The image recognition community has produced a number of data augmentation techniques that could very likely help. Some early work in domain transfer—training on data from a similar domain, but with more data available—has shown promising results[40]. Reinforcement learning can also be used where no data exists, but where a system for the learning agent to interact with can be setup.

A few selected algorithms seem like a good fit to solve some of the previous problems:

- Graph convolutional networks. Just as grid convolutional networks have been used for generating images[41], convolutional networks on graphs[42] could be used to generate game scenarios on graph representations.
- Autoencoders, Recurrent Networks and Generative Adversarial Networks. These networks are seeing widepsread use for image generation (see for example Ref. 43). LSTM recurrent networks have already been used for generating 2D platformer levels[25] and can probably be adapted to more difficult tasks. GANs have not yet been applied to videogames, but the success of autoencoders on generating, repairing and evaluating *Super Mario Bros.* levels[26] makes us confident that they will be beneficial. Additionally, the discriminator network of the GAN, can help identify unplayable content.
- One-shot and few-shot learning. There is a growing corpus of research on one-shot[44] and few-shot learning[45,46], where a classifiers must generalize after seeing a few (or only one) instances of a new category. These methods could help with learning to generate content starting from a few human authored examples.

We can combine these techniques with successful search-based and traditional

techniques. We could represent the content space indirectly as a generative grammar system. An evolutionary algorithm searches the space, and a neural network serves as the evaluation function.

4. Potential Impact

Procedural content generation has the potential to significantly reduce costs and development time, be it by assisting designers or by automatically generating the content. This is particularly marked in the development of sequels or *reskins*. A reskin is a common mobile game development practice where the company will release several games that look different but share the same code. If the *look* part (textures, maps, levels) is automatically generated, reskinning is almost free.

Also, the value of the final product is increased, by having a volume of content that is essentially infinite. This can be manifested be either regenerating the game world every time the player starts it (*SimCity*), to having the world be infinite for all practical purposes (*No Man's Sky*). In both cases, replayability is far greater than what could be achieved by traditional design processes.

An ability that can never be mimicked by human designers, is the ability to generate content tailored to each specific player, and to act instantaneously on implicit feedback by the player. Personalized content could be a game changer in the industry, the way that online gaming was, and virtual reality will soon be.

Finally, the most underdeveloped category of game content, derived content, can help foster a community around a game, increasing the time span during which the game is generating sales. These communities usually grow naturally around the biggest blockbusters, helped by marketing teams. However, in smaller studios, it might help to generate some content for these communities consumption (leaderboards, news about the in-game world) automatically, so as not to overtax limited human resources.

References

1. M. Hendrikx, S. Meijer, J. Van Der Velden and A. Iosup, Procedural content generation for games: A survey, *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)* **9**(1) (2013) p. 1.
2. J. Togelius, G. N. Yannakakis, K. O. Stanley and C. Browne, Search-based procedural content generation: A taxonomy and survey, *IEEE Transactions on Computational Intelligence and AI in Games* **3**(3) (2011) 172–186.
3. N. Shaker, J. Togelius and M. J. Nelson, *Procedural Content Generation in Games: A Textbook and an Overview of Current Research* (Springer, 2016).
4. G. Smith, *Procedural Content Generation: An Overview*, in *Game AI Pro 2: Collected Wisdom of Game AI Professionals* (CRC Press, 2015), ch. 40, pp. 501 – 518.
5. A. Summerville, S. Snodgrass, M. Guzdial, C. Holmgård, A. K. Hoover, A. Isaksen, A. Nealen and J. Togelius, Procedural content generation via machine learning (pcgml), *arXiv preprint arXiv:1702.00539* (2017).
6. R. van der Linden, R. Lopes and R. Bidarra, Procedural generation of dungeons, *IEEE Transactions on Computational Intelligence and AI in Games* **6**(1) (2014) 78–89.

7. N. Shaker, M. Nicolau, G. N. Yannakakis, J. Togelius and M. O'neill, Evolving levels for super mario bros using grammatical evolution, in *Computational Intelligence and Games (CIG), 2012 IEEE Conference on* IEEE2012, pp. 304–311.
8. Y. I. Parish and P. Müller, Procedural modeling of cities, in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques* ACM2001, pp. 301–308.
9. W. M. Reis, L. H. Lelis and Y. K. Gal, Human computation for procedural content generation in platform games, in *2015 IEEE Conference on Computational Intelligence and Games (CIG)*Aug 2015, pp. 99–106.
10. E. J. Hastings, R. K. Guha and K. O. Stanley, Automatic content generation in the galactic arms race video game, *IEEE Transactions on Computational Intelligence and AI in Games* **1**(4) (2009) 245–263.
11. E. J. Hastings, R. K. Guha and K. O. Stanley, Evolving content in the galactic arms race video game, in *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on* IEEE2009, pp. 241–248.
12. A. Martin, A. Lim, S. Colton and C. Browne, Evolving 3d buildings for the prototype video game subversion, in *European Conference on the Applications of Evolutionary Computation* Springer2010, pp. 111–120.
13. J. Togelius, R. De Nardi and S. Lucas, Making racing fun through player modeling and track evolution, in *Proceedings of the SAB Workshop on on Adaptive Approaches to Optimizing Player Satisfaction*2006.
14. J. Togelius, R. De Nardi and S. M. Lucas, Towards automatic personalised content creation for racing games, in *Computational Intelligence and Games, 2007. CIG 2007. IEEE Symposium on* IEEE2007, pp. 252–259.
15. M. Frade, F. F. de Vega and C. Cotta, Evolution of artificial terrains for video games based on accessibility, in *European Conference on the Applications of Evolutionary Computation* Springer2010, pp. 90–99.
16. J. Togelius, M. Preuss and G. N. Yannakakis, Towards multiobjective procedural map generation, in *Proceedings of the 2010 workshop on procedural content generation in games* ACM2010, p. 3.
17. J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelbäck and G. N. Yannakakis, Multiobjective exploration of the StarCraft map space, in *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on* IEEE2010, pp. 265–272.
18. D. Oranchak, Evolutionary algorithm for generation of entertaining shinro logic puzzles, in *European Conference on the Applications of Evolutionary Computation* Springer2010, pp. 181–190.
19. D. Ashlock, Automatic generation of game elements via evolution, in *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on* IEEE2010, pp. 289–296.
20. C. Pedersen, J. Togelius and G. N. Yannakakis, Modeling player experience in super mario bros, in *Computational Intelligence and Games, 2009. CIG 2009. IEEE Symposium on* IEEE2009, pp. 132–139.
21. N. Sorenson and P. Pasquier, Towards a generic framework for automated video game level creation, in *European Conference on the Applications of Evolutionary Computation* Springer2010, pp. 131–140.
22. V. Hom and J. Marks, Automatic design of balanced board games, in *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE)*2007, pp. 25–30.
23. J. Togelius and J. Schmidhuber, An experiment in automatic game design, in *Computational Intelligence and Games, 2008. CIG'08. IEEE Symposium On* IEEE2008, pp. 111–118.

24. C. B. Browne, *Automatic generation and evaluation of recombination games*, PhD thesis, Queensland University of Technology2008.
25. A. J. Summerville and M. Mateas, Super mario as a string: Platformer level generation via lstms, *DiGRA/FDG* (2016).
26. R. Jain, A. Isaksen, C. Holmgård and J. Togelius, Autoencoders for level generation, repair, and recognition, in *Proceedings of the ICCC Workshop on Computational Creativity and Games*2016.
27. I. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville and Y. Bengio, Generative adversarial nets, in *Advances in neural information processing systems*2014, pp. 2672–2680.
28. S. Snodgrass and S. Ontañón, Learning to generate video game maps using markov models, *IEEE Transactions on Computational Intelligence and AI in Games* **9**(4) (2017) 410–422.
29. A. Krizhevsky, I. Sutskever and G. E. Hinton, Imagenet classification with deep convolutional neural networks, in *Advances in neural information processing systems*2012, pp. 1097–1105.
30. G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath *et al.*, Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups, *IEEE Signal Processing Magazine* **29**(6) (2012) 82–97.
31. D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, Mastering the game of go with deep neural networks and tree search, *Nature* **529**(7587) (2016) 484–489.
32. D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, Mastering the game of go without human knowledge, *Nature* **550**(7676) (2017) p. 354.
33. A. Summerville, M. Guzdial, M. Mateas and M. O. Riedl, Learning player tailored content from observation: Platformer level generation from video traces using lstms, in *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*2016.
34. S. Snodgrass and S. Ontañón, Experiments in map generation using markov chains., in *FDG*2014.
35. S. Snodgrass and S. Ontañón, Controllable procedural content generation via constrained multi-dimensional markov chain sampling., in *IJCAI*2016, pp. 780–786.
36. S. Snodgrass and S. Ontañón, A hierarchical mdmc approach to 2d video game map generation, in *Eleventh Artificial Intelligence and Interactive Digital Entertainment Conference*2015.
37. S. Lee, A. Isaksen, C. Holmgård and J. Togelius, Predicting resource locations in game maps using deep convolutional neural networks, in *The Twelfth Annual AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment. AAAI*2016.
38. M. Guzdial and M. Riedl, Game level generation from gameplay videos, in *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*2016.
39. M. Guzdial, B. Harrison, B. Li and M. Riedl, Crowdsourcing open interactive narrative., in *FDG*2015.
40. S. Snodgrass and S. Ontañón, An approach to domain transfer in procedural content generation of two-dimensional videogame levels, in *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*2016.
41. A. Dosovitskiy, J. T. Springenberg and T. Brox, Learning to generate chairs with convolutional neural networks, in *Computer Vision and Pattern Recognition (CVPR), 2015 IEEE Conference on* IEEE2015, pp. 1538–1546.
42. T. N. Kipf and M. Welling, Semi-supervised classification with graph convolutional

networks, in *International Conference on Learning Representations (ICLR)*2017.
43. K. Gregor, I. Danihelka, A. Graves, D. Rezende and D. Wierstra, Draw: A recurrent neural network for image generation, in *International Conference on Machine Learning (ICML)*2015, pp. 1462–1471.
44. L. Fei-Fei, R. Fergus and P. Perona, One-shot learning of object categories, *IEEE transactions on pattern analysis and machine intelligence* **28**(4) (2006) 594–611.
45. S. Ravi and H. Larochelle, Optimization as a model for few-shot learning, in *International Conference on Learning Representations (ICLR)*2017.
46. J. Snell, K. Swersky and R. Zemel, Prototypical networks for few-shot learning, in *Advances in Neural Information Processing Systems*2017, pp. 4080–4090.