Applications of Metaheuristic Algorithms

Alexandru Ionascu, AIDC

January 2020

1 Introduction

This paper review will present highlights of important papers and implementations that provided significant improvements mostly in the computer industry using metaheuristic algorithms. The following presented works are in an arbitrary order and are selected among personal preferences, popularity, and accordance with the currently pursued specialization: artificial intelligence and distributed computing. In this context, popularity can mean mediatic popularity and large audiences, but also popular among highly specialized communities.

Metaheuristic algorithms applications have been proved useful in countless situations. Starting with 1986 metaheuristic algorithms notion have been widely used and recent surveys are showing a steady increase in popularity, peaking in 2015[7].

The large array of applications and use cases is due to the universality of optimization problems, thus every domain provides numerous applications for metaheuristic algorithms.

There are plenty of advanced optimizations methods nowadays and in practice, we usually need the single best one for our situation. Metaheuristics are still very popular in this competition, not only due to their relative simplicity but also for their capability of finding superior solutions for many optimization problems on popular datasets. This paper will select such cases which provided practicality and significant impact among the recent years in multiple subfields involving computing.

2 Paper Reviews

2.1 TPOT: A Tree-based Pipeline Optimization Tool for Automating Machine Learning[9]

2.1.1 Motivation

This is a very popular tool among the Kaggle[6] community. The Python implementation is also widely popular on Github and it's been proven very useful in real-world machine learning competitions. In one of the earlier releases, there

was a top 25% challenge using only genetic-programming-based TPOT, and the community has responded very positively in this regard, in some popular challenges reaching even top 10% on public leaderboards.

2.1.2 Description of the problem

To approach a machine learning problem, several prerequisites are enforced to achieve desired results. First, a good understanding of machine learning algorithms is required. Second, expert knowledge of the chosen problem and the domain-specific knowledge can be crucial. Thirdly, it requires at least intermediate-level knowledge of programming.

The presented paper addresses these prerequisites and makes it possible with none of them. More than that, it is based on the philosophy of not needing human intervention.

TPOT creates machine learning pipelines by using genetic programming to construct automatically Python programs.

2.1.3 Critical Review

The main purpose of the offered solution is to generate Python code based on the sci-kit-learn library [10] and XGBoot[1] integration exclusively. Almost all the end-user modules from sci-kit are being used and at the high level, the author distinguishes the following categories of operators:

- Supervised Classification Operators
- Feature Preprocessing Operators
- Feature Selection Operators

The idea is to create a multiple tree-based machine learning pipeline using the operators which they all behave in the same way: receive input data and output data. The main difference the author provides comparing to other similar approaches it's that there are no restrictions on the shape and the size of the pipeline. The Genetic Programming implementation is provided by another popular python framework, DEAP[4].

Overall the technical implementation is simple. The genetic programming part is standard, and the fact that scikit has a consistent module system and pipelining support it makes the implementation straightforward. Even scikit vector transformations are already done based on numpy. It's obvious that the code generation part is almost redundant since the automatic pipeline generation is the most powerful idea that the community adopted really quick.

2.1.4 Results

The authors ran 150 benchmarks. The comparisons were done between TPOT and random forest models with 500 trees, which is a fair comparison to a novice practitioner in machine learning. The maximum time to compute is 8 hours.

TPOT discovered statistically significantly better pipelines on 21 benchmarks, worse on 4, and on 125 no evidence.

The results are decent because the worse cases had 2-5% worse performance. It is no breakthrough but it almost guarantees a good baseline. However, the most important aspect is that the community proved to find useful competitions for this solution.

2.1.5 Limitations

The most critics in genetic programming consist of optimizing large populations leading to wasteful solutions. The authors cited some improvements, however, the improvements are marginal.

2.1.6 Open Problems

One idea would be meta-learning, in brief, it would use previous machine learning runs to predict the performance of the pipeline on a particular data set.

2.1.7 Conclusion

The main idea is that AutoML works up to a fairly good point. We don't have state of the art solutions for problems solved in a black box, but the automatic model generation has been positively tested on both benchmarks and community.

AutoML has proven at least in some particular cases more powerful than the average machine learning practitioner. The idea and the implementation is simple and straightforward, the results are sound, we are expecting improvements soon in this field.

2.2 Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning[11]

2.2.1 Motivation

This article became quickly very popular among machine learning open source communities. The selections are not arbitrary since it generated plenty of topics of discussion among the results of the findings. Nevertheless, it attracted plenty of open source contributors to the mentioned challenges, and also strongly encouraged adopting genetic algorithms in deep reinforcement learning and also applying new genetic algorithms for neuroevolution like we never have been using before.

2.2.2 Description of the problem

This paper will focus on comparing the performance of deep reinforcement learning algorithms with a genetic algorithm to train a neural network. The most

surprising fact is that along the recent years, gradient-based methods are de facto methods for such tasks. Everybody is expecting genetic algorithms to behave slower than a gradient-based approach, whether we talk about gradient calculation on an estimate.

2.2.3 Critical Review

The paper is approaching some of the popular Open AI environments which require agents to usually maximize a long term goal, usually in a video game or in a 3D mechanical simulation. The performance of the genetic algorithm should compare with state of the art deep reinforcement learning (DQN, Q-learning, A3C[8] from Google Deep Mind), policy gradient method and Evolution Strategies).

Comparing with other similar resource attempts the authors state that the genetic algorithm hasn't been put in enough computational power as now. The deep genetic algorithm is now capable to successfully evolve over 4 million parameters. The results confirm the claim that more computing power, especially CPU-only resources are leading to much better results than previously thought. But the surprising fact is that for some environments, the genetic algorithm reaches faster to near-optimal solutions as short as 10 generations.

2.2.4 Results

This is the core of the paper and a very dense area with different algorithms, environments, and results. The first section includes Atari video-game environments, both pixel-based environments, and simulations. Secondly, there is the humanoid locomotion challenge. In the first category, the genetic algorithm approach is considerably competitive, but in the second one, the state of the art methods perform much better.

Surprisingly, the convergence of the genetic algorithm was so fast that the authors believed that it was due to too much exploration and with enough resources, it would be a slightly different random search. And they took into considerations also random search. And another interesting finding is that the state of the art solutions didn't perform better in all the cases comparing to the random search, but the genetic algorithm did.

2.2.5 Open Problems

The biggest improvement for deep neural network training for a genetic algorithm mentioned in the paper is Novelty Search. The authors expect other methods too to improve the deep neural network training dramatically.

2.2.6 Conclusion

This paper showed promising results regarding non-gradient deep reinforcement learning methods using genetic algorithms instead. One reason might be that not all the problems have a search space suitable for a gradient approach. Even

though it's easy for a gradient method to escape a local optimum (most of the results are taken into account the momentum and the adaptive learning rate) they are outperformed by genetic algorithms and not only, in some cases even by random search.

Novelty Search Genetic Algorithm was proven to perform extremely well and we definitely expect to see more enhancement algorithms for a better genetic process. Now we can confidently stop overlooking non-gradient methods that are widely used and consider other alternatives for deep neural network training.

3 Relevant Papers

- *Multi-itinerary optimization as cloud service*[2]

  In this article, the key metaheuristic is an enhanced variant of Adaptive Large Neighbourhood Search metaheuristic applied for solving a time-window constrained traveling salesman problem in a public web service offered by Microsoft called Multi Itinerary Optimization. The results are up to 17% better to the current state of the art algorithm and it's performing way better than the mixed-integer linear programming commercial solvers.

- *Parallel machine scheduling with dynamic resource allocation via a master-slave genetic algorithm: Parallel machine scheduling with dynamic resource allocation* [5]

  A relatively recent article and an unusual idea. The authors present a classical formulation of an NP-problem of task scheduling for parallel tasks. The results are promising in comparison with other good performing metaheuristics.

- *Cloudde: A heterogeneous differential evolution algorithm and its distributed cloud version* [12]

  The paper presents a double layered master-slave architecture. The master controls the heterogeneous populations. Based on the size of the population, the resources are allocated respectively.

  The key point about using this approach is that the populations are sorted based on their average fitness. Each individual has a chance of migrating in any of the superior populations. By doing this, the lower performing populations based by the initial parameter configuration don't use more computing resources than the better performing ones.

- *On the Bounds of Function Approximations* [3] An article by Alexa Research showed that genetic algorithms were the best algorithm-selection for Approximate Architecture Search Problem (a-ASP).

4 Conclusion

It's difficult to synthesize the applications of a subfield of over 35 years of constantly increasing interest in publications. We have only given a few applications of metaeuristics published in recent years, prioritizing the popularity and impact among the communities of practitioners of the specific fields. The in-depth analysis focused on machine learning tasks, and in the latter examples, we've shown applications in distributed systems, parallel computing, and large scale web services for successfully solving NP tasks.

Even though we have more articles published in metaheuristic than the decades before we still witness impactful work from both Acamedia and industry, and we have more directions to explore than we had before.

References

[1] Tianqi Chen and Carlos Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.

[2] Alexandru Cristian, Luke Marshall, Mihai Negrea, Flavius Stoichescu, Peiwei Cao, and Ishai Menache. Multi-itinerary optimization as cloud service (industrial paper). In *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '19, page 279–288, New York, NY, USA, 2019. Association for Computing Machinery.

[3] Adrian de Wynter. On the Bounds of Function Approximations. *arXiv e-prints*, page arXiv:1908.09942, Aug 2019.

[4] Félix-Antoine Fortin, François-Michel De Rainville, Marc-André Gardner, Marc Parizeau, and Christian Gagné. DEAP: Evolutionary algorithms made easy. *Journal of Machine Learning Research*, 13:2171–2175, jul 2012.

[5] Yaping Fu, Guangdong Tian, Zhiwu Li, and Zhenling Wang. Parallel machine scheduling with dynamic resource allocation via a master-slave genetic algorithm: Parallel machine scheduling with dynamic resource allocation. *IEEJ Transactions on Electrical and Electronic Engineering*, 13, 01 2018.

[6] Alphabet Inc. *www.kaggle.com*.

[7] Shi Cheng Yuhui Shi Kashif Hussain, Mohd Najib Mohd Salleh. Metaheuristic research: a comprehensive survey. *Springer Science Business Media B.V., part of Springer Nature*, 2018.

[8] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu.

Asynchronous Methods for Deep Reinforcement Learning. *arXiv e-prints*, page arXiv:1602.01783, Feb 2016.

[9] Randal S. Olson and Jason H. Moore. Tpot: A tree-based pipeline optimization tool for automating machine learning. pages 151–160, 2019.

[10] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[11] Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, Joel Lehman, Kenneth O. Stanley, and Jeff Clune. Deep Neuroevolution: Genetic Algorithms Are a Competitive Alternative for Training Deep Neural Networks for Reinforcement Learning. *arXiv e-prints*, page arXiv:1712.06567, Dec 2017.

[12] Z. Zhan, X. Liu, H. Zhang, Z. Yu, J. Weng, Y. Li, T. Gu, and J. Zhang. Cloudde: A heterogeneous differential evolution algorithm and its distributed cloud version. *IEEE Transactions on Parallel and Distributed Systems*, 28(3):704–716, March 2017.