Towards optimisers that 'Keep Learning'

Emma Hart
Edinburgh Napier University
Edinburgh, Scotland, UK
e.hart@napier.ac.uk

Ian Miguel
University of St Andrews
St Andrews, Scotland, UK
ijm@st-andrews.ac.uk

Christopher Stone
University of St Andrews
St Andrews, Scotland, UK
cls29@st-andrews.ac.uk

Quentin Renau
Edinburgh Napier University
Edinburgh, Scotland, UK
q.renau@napier.ac.uk

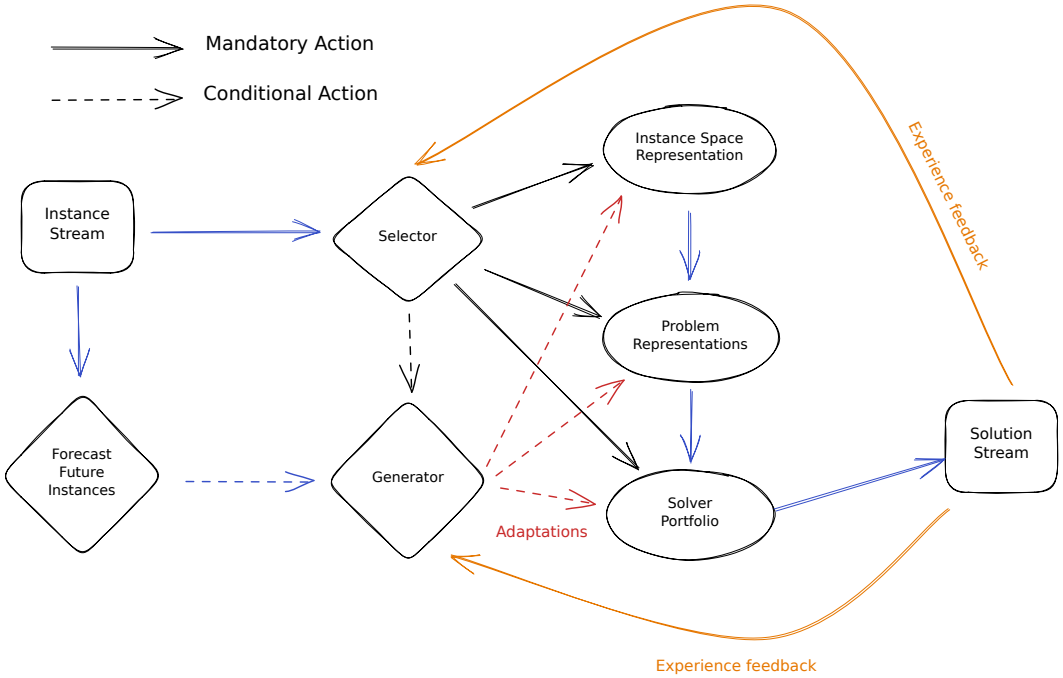

Figure 1: Overview of an optimisation system that Keeps Learning

ABSTRACT

We consider optimisation in the context of the need to apply an optimiser to a continual stream of instances from one or more domains, and consider how such a system might '*keep learning*': by drawing on past experience to improve performance and learning how to both predict and react to instance and/or domain drift.

GECCO '23 Companion, July 15–19, 2023, Lisbon, Portugal

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/3583133.3596344

CCS CONCEPTS

• **Computing methodologies** → **Search methodologies**.

KEYWORDS

Optimisation, continual learning, transfer-learning

ACM Reference Format:
Emma Hart, Ian Miguel, Christopher Stone, and Quentin Renau. 2023. Towards optimisers that 'Keep Learning'. In *Genetic and Evolutionary Computation Conference Companion (GECCO '23 Companion), July 15–19, 2023, Lisbon, Portugal.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3583133.3596344

1 MOTIVATION

Combinatorial problems are ubiquitous across many sectors where delivering optimised solutions can lead to considerable economic

benefits in many fields, e.g. in logistics, manufacturing, and scheduling. In many scenarios, optimisation algorithms are repeatedly applied to instances from a domain. For example, in routing applications such as online delivery, routes might be produced many times a day in response to demand, while in factories, processes on machines need to be repeatedly scheduled based on customer orders. In these types of domain, the frequency with which instances occur (and need to be solved) can also vary widely across domains, ranging from many times a day to longer timescales. We argue that optimisation in the context of continually arriving instances (i.e. streaming data) offers a unique opportunity for an optimiser to *learn*:

- Over time, a rich history of data is accumulated, comprising of at minimum the instance data, the solution and the algorithm used to solve it. Therefore this information could be mined and exploited in order to improve existing solvers
- The characteristics of the stream of instances are likely to change over time, given that the world in which we operate is intrinsically dynamic. Optimisers could therefore autonomously learn to recognise change (whether it is gradual or sudden) and adapt accordingly

If optimisers are not capable of learning after deployment, then in the best case this leads to systems that deliver sub-optimal performance, while at worst, systems that are completely unfit for purpose. Therefore, in this position paper we discuss the questions of *when* to learn, *what* to learn and address the question of *how* to learn in the context of optimisers applied to a continual stream of instances. The discussion is separated into two parts. In the first, we discuss how optimisers might learn from experience. We follow this with a discussion of how optimisers can recognise change in an instance stream and react accordingly.

2 LEARNING FROM EXPERIENCE

In the machine-learning community, Silver [8] noted a decade ago that "*.. it is now appropriate for the AI community to move beyond learning algorithms to more seriously consider the nature of systems that are capable of learning over a life time*". While the ML community has embraced this challenge, for example via *continual learning* and *transfer learning* [7] — learning a new task through the transfer of knowledge from a related task[1] that has already been learned — it is much less common in optimisation. We suggest that knowledge can be transferred at multiple levels. For instance, an algorithm configuration; an initial starting solution (i.e. warm-starting [4]); partial solutions (i.e. fragments of a good solution); at the *operator* level (e.g. specific mutation operators or sequences of operator applications). While algorithm-configuration methods have received some attention in the literature, population-based meta-heuristic optimiser tend to start a search from scratch each time a new instance appears, assuming zero prior knowledge about the task at hand. Recently, there has been growing interest in the field of evolutionary transfer optimisation (ETO) [12]: a paradigm that integrates meta-heuristic solvers with knowledge learning and transfer methods across related instances *and* domains[2] to either improve efficiency or performance. Part of the challenge is of course deciding what information to transfer. Another pertinent question is 'when is an instance sufficiently like a previous one that information can usefully be transferred'. Often this is tackled by extracting human-designed features from an instance (either based on the instance data or its fitness landscape).

Furthermore, meta-heuristic approaches expect handcrafted representations tailored to the problem at hand, which makes them unable to learn about the effects of different representations. However, novel systems [2, 6] can reformulate a domain into different representations and select one automatically. This means it is now possible to learn from previous selections, which should inform future decisions. These tools can tap into a large variety of general systematic solvers, such as constraint and SAT solvers, Linear Programming solvers, and Satisfiability Modulo Theory solvers by translating the problem to the appropriate input language. Given the right reformulation and configuration, these solvers can potentially tackle combinatorial problems in a wide range of domains. We suggest that this is better tackled by understanding similarities/differences from multiple points of view by looking at the interactions between instances, representations and solvers rather than just features. Clearly, there is much potential for future work in this area, focusing on the what, how and when of choosing to transfer information.

3 LEARNING TO ADAPT

We suggest that it is essential that optimisers that face continual streams of instances to optimise are able to adapt to unknown data. On the one hand, a system should be able to deal with instances from a domain that lies in a region of an instance-space [10] that is outwith the known footprint of an algorithm or portfolio: here we should also be able to distinguish between *drift* (the instance lies close to known regions) and *surprise* (the instance lies in a completely new region). On the other hand, it should ideally also be able to deal with instances from a completely different domain.

This has two challenges. The first is related to how to detect whether an instance lies in an unknown region or in a new domain, and secondly how to deal with these scenarios. With respect to the former, instance-space analysis techniques [10] offer a visual way to detect drift or surprise, but have some shortcomings [9]. Anomaly-detection methods derived from the ML community may also have a role to play here, particularly if they can quantify the magnitude of change [5]. Detecting domain drift is more challenging: a minimal requirement for this would be to have a common representation that can be used across multiple domains. One way to facilitate this is via the use of high-level constraint specification languages such as Essence [3], which allows a combinatorial (optimisation) problem to be specified at a level of abstraction above that at which modelling decisions are made, and potentially enables one domain to be transformed into another at this level. Change at a lower level might be dealt with by parameter tuning for a given solver. On the other hand, significant change may involve updating a portfolio or automated-algorithm generation techniques [11]. Going even further, a system should also be able to *predict* change, i.e. be proactive rather than reactive. If this can be achieved (e.g. by tracking trajectories through instance-space), then this can

[1] We interpret the term 'task' to apply either a new instance, a new objective function or a new domain

[2] typically referred to as multi-task optimisation in the literature

be tackled by first generating new instances in predicted regions of space [1], and secondly, using automated-algorithm generation techniques as suggested above.

4 CONCLUSION

In summary, this short position paper suggests that optimisation systems should:

- Learn from experience (therefore improving over time), including learning across domains
- React seamlessly to drift and surprise with respect to a stream in instances from a domain
- Be able to handle completely new tasks from unseen domains

There are many avenues of current research within both the meta-heuristic and constraint-based programming community that tackle pieces of this jigsaw that can be drawn upon and adapted to create the utopian optimisation system we envisage. It also seems clear that this needs to be an interdisciplinary effort, particularly drawing in techniques from the ML community in continual and multi-task learning and anomaly detection.

ACKNOWLEDGEMENTS

The authors are funded by the EPSRC 'Keep-Learning' project: EP/V026534/1 and EP/V027182/1

REFERENCES

[1] Özgür Akgün, Nguyen Dang, Ian Miguel, András Z Salamon, and Christopher Stone. 2019. Instance generation via generator instances. In *Principles and Practice of Constraint Programming: 25th International Conference, CP 2019, Stamford, CT, USA, September 30–October 4, 2019, Proceedings 25*. Springer, 3–19.

[2] Özgür Akgün, Alan M Frisch, Ian P Gent, Christopher Jefferson, Ian Miguel, and Peter Nightingale. 2022. Conjure: Automatic generation of constraint models from problem specifications. *Artificial Intelligence* 310 (2022), 103751.

[3] Alan M Frisch, Warwick Harvey, Chris Jefferson, Bernadette Martínez-Hernández, and Ian Miguel. 2008. Essence: A constraint language for specifying combinatorial problems. *Constraints* 13 (2008), 268–306.

[4] Anja Jankovic, Diederick Vermetten, Ana Kostovska, Jacob de Nobel, Tome Eftimov, and Carola Doerr. 2022. Trajectory-based algorithm selection with warm-starting. In *2022 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 1–8.

[5] Tung Kieu, Bin Yang, Chenjuan Guo, and Christian S Jensen. 2019. Outlier Detection for Time Series with Recurrent Autoencoder Ensembles.. In *IJCAI*. 2725–2732.

[6] Peter Nightingale, Özgür Akgün, Ian P Gent, Christopher Jefferson, Ian Miguel, and Patrick Spracklen. 2017. Automatically improving constraint models in Savile Row. *Artificial Intelligence* 251 (2017), 35–61.

[7] German I Parisi, Ronald Kemker, Jose L Part, Christopher Kanan, and Stefan Wermter. 2019. Continual lifelong learning with neural networks: A review. *Neural networks* 113 (2019), 54–71.

[8] Daniel L Silver, Qiang Yang, and Lianghao Li. 2013. Lifelong machine learning systems: Beyond learning algorithms. In *2013 AAAI spring symposium series*.

[9] Kevin Sim and Emma Hart. 2022. Evolutionary Approaches to Improving the Layouts of Instance-Spaces. In *Parallel Problem Solving from Nature–PPSN XVII: 17th International Conference, PPSN 2022, Dortmund, Germany, September 10–14, 2022, Proceedings, Part I*. Springer, 207–219.

[10] Kate Smith-Miles and Mario Andrés Muñoz. 2021. Instance space analysis for algorithm testing: Methodology and software tools. *Comput. Surveys* (2021).

[11] Thomas Stützle and Manuel López-Ibáñez. 2019. Automated design of metaheuristic algorithms. *Handbook of metaheuristics* (2019), 541–579.

[12] Kay Chen Tan, Liang Feng, and Min Jiang. 2021. Evolutionary Transfer Optimization - A New Frontier in Evolutionary Computation Research. *IEEE Computational Intelligence Magazine* 16, 1 (2021), 22–33. https://doi.org/10.1109/MCI.2020.3039066