

Data Science Meets Optimization

Patrick De Causmaecker

1 Introduction

Data science and optimisation have evolved separately over several decades. In [16], Tony Hey and his co-authors, inspired by the late, Turing award winning, Jim Gray picture data science as a fourth paradigm in scientific evolution. After the empirical branch, the theoretical and the computational branch, data exploration comes to the scene. Historically, this evolution spreads over literary thousand years and most of research domains can be traced back a long time in history. An interesting overview on combinatorial optimisation is [23]. Interesting cases are e.g. the Steiner problem [1, 4] and Kepler's conjecture [15].

Here we will review some recent developments in the application of data science to support heuristics for combinatorial optimisation problems as well as the use of heuristics in a data science context.

2 Data Science for Optimisation

Optimisation is the discipline of finding values of variables that optimise some goal function(s) within a given domain. The description of the problem is typically not more than a couple of pages long. Its complexity status may or may not be known. In practical situations, the problem is not in the complexity status, but in the expected execution time for a specific instance.

Data science may enter in various ways. The difficulty of problems often strongly depends on the properties of the problem instance. Producing optimal solutions in

P. De Causmaecker (✉)

CODES, Department of Computer Science, KU Leuven/KULAK,
E. Sabbelaan 53, 8500 Kortrijk, Belgium
e-mail: patrick.decausmaecker@kuleuven.be

a guaranteed amount of computation time is often not possible. The problem shifts from finding the optimal solution towards finding an as good as possible solution in an acceptable amount of time for most of the problem instances that will be thrown at the system.

2.1 *DFO Example: Using Data Science in Algorithm Construction*

A first example with remarkable successes in recent years is automated algorithm construction. Automated algorithm construction is one possibility in a set of automated mechanisms allowing data to be used for optimisation. Available are a set of problem instances and a formal description of the problem. The aim is to produce an algorithm that solves the problem as efficiently as possible. One could argue that techniques such as algorithm selection and algorithm tuning are in fact algorithm construction techniques. In algorithm selection, a set of algorithms for the problem is defined and the aim of the automated construction phase is limited to learning a predictor from the data that selects the best algorithm to solve a given problem. In algorithm tuning, one algorithm is given with a number of discrete and continuous parameters. The aim of the automated phase is to find the best setting of the parameters for the expected distribution of the problem instances of which the data is supposed to be a representative sample.

An obvious argument against is that in selection, as well as in tuning, complete algorithms have been designed and only a limited number of decisions are left to the automated construction module. In the other extreme, no constraints except the available operations at the machine or programming language level should exist.

As an example we discuss the approach in [2] for multi objective evolutionary algorithm (MOEA) construction. The authors present a general conceptual view of an MOEA, and prior to the automated approach, demonstrate how a number of well known MOEA's from literature can be described. MOEA's are seen as combinations of lower-level components such as preference relations and archives. The resulting framework extends the number of algorithms that can be instantiated as well as the number of MOEA approaches that can be designed. Given this large design space, an efficient navigator is needed to find the best combination of components for a given problem. The authors use the parameter tuning package *irace*[20] for this purpose. *irace* accepts discrete and continuous parameters and can hence be configured as a selector of components provided a specific pattern is given. *irace* then only has to decide which component to insert where. The authors design new databases of instances for several problems and study the behaviour of the constructor. The same problem with different instances turns out to lead to significantly different algorithms. The constructed MOEA's outperform existing algorithms, even after these existing algorithms have been tuned by *irace*.

This example uses expert human knowledge formalised in an algorithm template and a black box analysis of a representative data set of instances. Some other examples are [3, 6, 18, 19, 21, 24]. The algorithm components in this approach are general and not problem specific. In the next section we discuss an approach that allows algorithm designers to insert knowledge about the problem. Optimisation is not only linked to data but also to expert knowledge.

2.2 DFO Example: Using Data Science While Engineering an Algorithm

Consider the problem of designing a multi-neighbourhood local search algorithm. Our running example will be a vehicle routing problem (VRP). A multi-neighbourhood local search algorithm is an algorithm that moves from one acceptable solution for the VRP to another according to a number of neighbourhoods. The local search could simply be hill climbing or it could employ a mechanism allowing it to escape from local optima. Selection of a neighbourhood may be (weighted) random selection or any intelligent, machine learning based, mechanism. A specific neighbourhood may reflect expert understanding of the problem domain or design experience. Designers will come up with a multitude of possible neighbourhoods. The question that remains is which neighbourhoods to actually use.

This is an algorithm construction problem as we saw in Sect. 2.1, with the weights as parameters. Another option is a hyper-heuristic approach [5] where the weights evolve during the search guided by machine learning. All these are black box approaches. The present example shows how to open the box.

The target is to offer support to algorithm designers. They will experiment with neighbourhood combinations on a set of sample problems. The information retrieved will be fragmented and no *why-questions* will be answered.

To allow observation of neighbourhood behaviour during the search [9] uses code inserts to register neighbourhood behaviour. Observables are goal function values, times taken, Evolution of these observables for each neighbourhood in the course of the search is stored. Inspection of the resulting file may a.o. reveal varying performance of neighbourhoods during the search.

The data plotted against time may not offer the full picture. Local search algorithms often wander around in extended valleys before discovering higher hills. A few observations in such a region may be sufficient to identify effective neighbourhoods. More interesting information comes when the goal function approaches its optimum.

The authors of [9] introduce the value of the goal function as the basis of a plot. This goal function value is taken from a, conveniently discretised, finite interval.

The collected data reflects the events during one run. The following features on each neighbourhood N include number of times used, number and amount of improvements, no effects, worsening. Each of these features is plotted against the

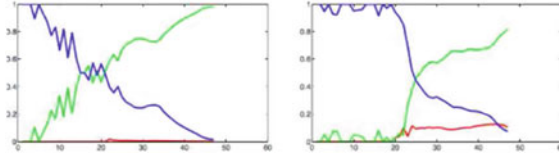


Fig. 1 Visualization of logged numbers of improvement, no effect and worsening for two neighbourhoods. The x-axis represents solution quality (the larger the value, the better the corresponding solution is). The y-axis represents values of the three observables

goal function value taken from a VNS algorithm. After dividing the goal function range interval into regions, [9] collects the quantities in a vector characterising each neighbourhood. See Fig. 1 for an example.

The procedure described now serves two purposes. It allows visualisation of neighbourhood behaviour for algorithm developers to select a diverse set of limited size or to identify properties of missing neighbourhoods. It allows analysing an existing algorithm automatically. If one supposes that neighbourhoods with similar profiles behave similarly, these characteristics may allow clustering neighbourhoods and selecting one from each cluster. This may lead to more efficient tuning possibilities.

In [9], such a study was performed. A set of 42 neighbourhoods was reduced to 9 and it was shown that the result of applying *iraces* significantly outperformed the 42-neighbourhood version.

3 Optimisation for Data Science

3.1 OFD Example: Community Detection in Graphs

Large graphs have many applications in network analysis. Examples are records of cell phone calls, e-mail connections, influencer graphs in social media, The scales of these networks vary from rather small (up to 1000 nodes) over medium (say 50000 nodes) up to very large graphs ($> 10^6$ nodes). For a review see [13, 14].

One subject of interest in such graphs is the detection of communities. Informally, communities are collections of nodes that are more connected to each other than to nodes outside the community. One way to model communities is by introducing objective functions reflecting the quality of a partition of the set of nodes. The problem then reduces to an optimisation problem finding the best partition given an objective function. An interesting extension is when the network is evolving as reflected by an evolution in the community structure. Although evolution could in principle discontinuously impact the community structure, this is mostly not the case in reality nor what one wants in the mathematical models. Systems relying on community structure (think of an advisory system for public relations support) would probably

not be helped by erratically changing the viewed community structure. This adds a second criterion to the optimisation problem, namely that of not changing abruptly. [7, 8]. We will refer to this problem as the Dynamic Community Problem (DCP).

DCP has been studied as a weighted optimisation problem with two goal functions called snapshot cost (SC) reflecting the absolute quality of the present structure and the temporal cost (TC) reflecting the amount of change [17]:

$$cost = \alpha.SC + (1 - \alpha).TC$$

The solvers based on this cost function typically depend on a rather large number of parameters and decisions. From an application point of view, an appropriate measure for the community quality SC is selected from a large number of possibilities from clustering theory. The evolution in the structure must be properly modeled in TC and the parameter α needs to be determined. These add to the parameters of the optimisation algorithm.

DCP is in essence situated in data science, the data being the interactions between the nodes of which the graph is a representation. When an optimisation approach is selected, parameter tuning, algorithm configuration and construction as discussed in Sect. 2.1 are needed.

As an example, we now discuss [12] in some detail. In this paper, DCP is recast as a multi-objective optimisation problem effectively getting rid of the weight α . Four versions of the algorithm are investigated, each using a different definition for SC . TC is computed through an expression from information theory measuring the similarity between community partitions. The algorithm is based on NGSA-II [10], and is taken from a *Matlab* library. The remaining parameters in the algorithm are linked to the genetic algorithm and the representation. Although the authors mention the possibility to tune these automatically, a trial and error approach is selected.

The experiments are performed on a number of synthetic data sets and on two data sets taken from real world cases. These data sets allow comparing against other algorithms, and it is shown that the algorithm outperforms existing ones. The sizes of the resulting graph range between 10^2 and 10^3 and must thus be considered small. The study is very similar to classical optimisation studies. An interesting analysis concerns complexity. The authors show that the complexity of the algorithm can be expressed as a function of the number of generations, the size of the population and the dimensions of the graph as

$$O(g.p.\log(p) \times (n.\log(n) + m))$$

where g is the number of generations, p is the size of the population, n is the number of nodes and m is the number of edges in the graph. Of course, the number of generations as well as the population size needed for acceptable results will vary with the size of the graph. The authors study the scalability of the method experimentally for numbers of nodes ranging from 128 to 4096 and numbers of edges ranging from 2938 to 65256. This allows quantifying these relationships to some extent. The authors conclude that a tradeoff between computing time and error gap is necessary.

So although the $n \log(n)$ dependency is very promising and should allow handling much larger graphs, the hidden dependency in the required number of iterations and population size together with the rather large constant required by the genetic algorithm are limiting factors.

The example discussed illustrates how a standard optimisation approach could be used to solve a problem from data science. The low complexity of metaheuristic approaches holds a promise for obtaining good solutions for large graphs.

4 Conclusions and Future Work

Data science supporting optimisation was illustrated through an example of algorithm construction 2.1 and an attempt to support algorithm design 2.2.

The construction case is the most general case of algorithm configuration and tuning which are generalisations of algorithm selection. The example relied on a framework for the design of a multi objective algorithm and specialised in selecting its components automatically. The link with data science is inherent in real world problems. Even when using exact algorithms, the parameter setting strongly influences computation time. When heuristics are used, it is even more important as the heuristics are often known to work well on a limited set of problem instances only. A representative sample of the expected set is the starting point for automated algorithm construction.

Another link was highlighted when discussing automated support for algorithm design. Again a framework was selected, represented by a local search approach. The main design issue is the selection of appropriate neighbourhoods relying on experience and intuition of designers. The support comes in when trial versions of the algorithm are run using the neighbourhoods. By observing this process, characteristic of the neighbourhoods emerged, suggesting good and worse sets. The example was taken one step further when the neighbourhoods were clustered automatically and the algorithm tuner was shown to profit from this information by producing better parameter settings.

Optimisation for data science was discussed in Sect. 3.1. The example given was a community detection algorithm for large graphs. By modelling the problem as a multi-objective optimisation problem, an off-the-shelf genetic algorithm for multi-objective optimisation could be used. The implementation in *Matlab* outperformed existing methods. An interesting complexity analysis revealed good asymptotic behaviour if one does not take the size of the population into account.

These examples suggest a number of possibilities for future research. Template based algorithm construction is an example of how human knowledge and data can be combined in a methodology that semi-automatically produces optimisation algorithms. The human knowledge is expressed through the template and the creation of components. The data is taken into account by the automatic configuration tool *iracethat* essentially runs the algorithm against a representative set of problem instances. Similar ideas are used in hyperheuristics [5] where human knowledge and

intuition are introduced through so called low level heuristics which are set to use by a sophisticated optimisation algorithm, often a genetic algorithm. A challenge for these lines of research is formalisation of the specification. As the example illustrates, and as can be seen in hyperheuristics literature, specification of components or low level heuristics still requires high level expertise. A domain expert cannot be expected to master the kind of sophistication needed to write efficient code for those algorithm components. A formal definition or a declarative language for specifying this information could be an important step forward to take this methodology to practice [11].

The design support for analysing the neighbourhoods in the local search algorithm could speed up the task of specialist algorithm designers. The sound mathematical basis used in the realisation of the characteristics and in the clustering exercise have presently been implemented in R [9]. Concerning analysing the behaviour of neighbourhoods, an interesting follow up research could be about interaction between neighbourhoods as exemplified by [22].

The optimisation for data science example concentrates on a genetic algorithm. The complexity of the algorithm depends on the number of generations and the size of the population. The accuracy of the result depends on the same parameters so that a best compromise needs to be sought. It may be interesting to investigate a local search approach which often leads to more efficient coding schemes, requires less involved data structures and takes more advantage of delta evaluation of the goal function. The question of the asymptotic behaviour needs an answer when looking at larger graphs.

Finally, all these examples were inspired by problems from practice. In the history of optimisation, many insights have emerged from theoretical considerations. Less theoretical effort has been spent on algorithm selection, configuration or construction. We are thinking of a hypothetical setting where the instance distribution is described analytically. Questions could be on a specific algorithm template, e.g. for a traveling salesperson problem when the cities live on a given structure or on the asymptotic behaviour of a community detection algorithm in terms of execution time as well as solution quality.

These and other questions could play a similar role as the complexity studies on classic optimisation problems and could help us understand the issues of DFO and OFD better.

References

1. Fermat-Torricelli problem. https://www.encyclopediaofmath.org/index.php/Fermat-Torricelli_problem. Accessed 20 May 2017
2. Bezerra, L.C., López-Ibáñez, M., Stützle, T.: Automatic component-wise design of multiobjective evolutionary algorithms. *IEEE Trans. Evol. Comput.* **20**(3), 403–417 (2016)
3. Bleuler, S., Laumanns, M., Thiele, L., Zitzler, E.: Pysaa platform and programming language independent interface for search algorithms. In: *International Conference on Evolutionary Multi-Criterion Optimization*, pp. 494–508. Springer, Berlin Heidelberg (2003)

4. Brazil, M., Graham, R.L., Thomas, D.A., Zachariasen, M.: On the history of the euclidean steiner tree problem. *Archi. Hist. Exact Sci.* **68**(3), 327–354 (2014)
5. Burke, E.K., Gendreau, M., Hyde, M., Kendall, G., Ochoa, G., Özcan, E., Qu, R. Hyper-heuristics: a survey of the state of the art. *J. Op. Res. Soc.* (2013) (to appear)
6. Burke, E.K., Hyde, M.R., Kendall, G., Ochoa, G., Ozcan, E., Woodward, J.R.: Exploring hyper-heuristic methodologies with genetic programming. In: *Computational Intelligence*, pp. 177–201. Springer, Berlin Heidelberg (2009)
7. Chakrabarti, D., Kumar, R., Tomkins, A.: Evolutionary clustering. In: *Proceedings of the 12th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '06*, pp. 554–560. ACM, New York, NY, USA (2006)
8. Chi, Y., Song, X., Zhou, D., Hino, K., Tseng, B.L.: On evolutionary spectral clustering. *ACM Trans. Knowl. Discov. Data*, **3**(4), 17:1–17:30, December (2009)
9. Dang, N.T.T., De Causmaecker, P.: Characterization of neighborhood behaviours in a multi-neighborhood local search algorithm. In: *International Conference on Learning and Intelligent Optimization*, pp. 234–239. Springer International Publishing (2016)
10. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T.A.M.T.: A fast and elitist multiobjective genetic algorithm: Nsga-ii. *IEEE Trans. Evol. Comput.* **6**(2), 182–197 (2002)
11. Devriendt, J., De Causmaecker, P., Denecker, M.: Transforming constraint programs to input for local search. In: *The Fourteenth International Workshop on Constraint Modelling and Reformulation*, pp. 1–16 (2015)
12. Folino, F., Pizzuti, C.: An evolutionary multiobjective approach for community discovery in dynamic networks. *IEEE Trans. Knowl. Data Eng.* **26**(8), 1838–1852 (2014)
13. Fortunato, S.: Community detection in graphs. *Phys. Rep.* **486**(3), 75–174 (2010)
14. Fortunato, S., Hric, D.: Community detection in networks: a user guide. *Phys. Rep.* **659**, 1–44 (2016)
15. Hales, T., Adams, M., Bauer, G., Dang, T.D., Harrison, J., Le Truong, H., Kaliszyk, C., Magron, V., McLaughlin, S., Nguyen, T.T., Nguyen, Q.T., Nipkow, T., Obua, S., Pleso, J., Rute, J., Solovyev, A., Hoai Thi Ta, A., Tran, T.N., Thi Trieu, D., Urban, J., Khac Vu, K., Zumkeller, R.: A formal proof of the Kepler conjecture. *ArXiv e-prints*, January (2015)
16. Hey, T., Tansley, S., Tolle, K.M., et al.: The fourth paradigm: data-intensive scientific discovery (2009)
17. Kim, M.-S., Han, J.: A particle-and-density based evolutionary clustering method for dynamic networks. *Proc. VLDB Endow.* **2**(1), 622–633 (2009)
18. Krasnogor, N., Smith, J.: A tutorial for competent memetic algorithms: model, taxonomy, and design issues. *IEEE Trans. Evol. Comput.* **9**(5), 474–488 (2005)
19. Liefvooghe, A., Jourdan, L., Talbi, E.-G.: A software framework based on a conceptual unified model for evolutionary multiobjective optimization: Paradiseo-moeo. *Eur. J. Op. Res.* **209**(2), 104–112 (2011)
20. López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L.P., Birattari, M., Stützle, T.: Iterated racing for automatic algorithm configuration. The irace package. *Op. Res. Persp.* **3**, 43–58 (2016)
21. Mascia, F., López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T.: Grammar-based generation of stochastic local search heuristics through automatic algorithm configuration tools. *Comput. Op. Res.* **51**, 190–199 (2014)
22. Misir, M., Verbeeck, K., De Causmaecker, P., Berghe, G.V.: An intelligent hyper-heuristic framework for chesc 2011. In: *Learning and Intelligent Optimization*, pp. 461–466. Springer, Berlin Heidelberg (2012)
23. Schrijver, A.: On the history of combinatorial optimization (till 1960). *Handb. Op. Res. Manag. Sci.* **12**, 1–68 (2005)
24. Srour, A., De Causmaecker, P.: Towards automatic design of adaptive evolutionary algorithms (2017)

Optimization and Decision Science: Methodologies and Applications

ODS, Sorrento, Italy, September 4-7, 2017

Sforza, A.; Sterle, C. (Eds.)

2017, XVI, 641 p. 99 illus., 52 illus. in color., Hardcover

ISBN: 978-3-319-67307-3