A causal framework for understanding optimisation algorithms

Alberto Franzin[0000−0002−4066−0375] and Thomas Stützle[0000−0002−5820−0473]

IRIDIA, Université Libre de Bruxelles (ULB), Brussels, Belgium
{afranzin,stuetzle}@ulb.ac.be

1 Introduction

Over the last decades, plenty of exact and non-exact methods have been proposed to tackle NP-hard optimisation problems. Despite the wide success of these methods, however, understanding their behaviour is still an open research question of fundamental interest [2]. Stochastic Local Search (SLS) algorithms mostly provide no guarantees on the running time or the quality of the solution returned [4]. Theoretical analyses of SLS behaviour are limited to specific cases or unrealistic assumptions such as infinite running time. The internal behaviour of branch-and-bound, the basis for state-of-the-art exact algorithms, is chaotic, and therefore it is almost impossible to predict the performance of a solver on a problem instance. Yet, all these algorithms are routinely used with success in countless applications. Thus, the gap between theoretical analyses and practical results is still very large.

In recent years, however, the adoption of rigorous experimental practices led many researchers to develop mathematical and statistical tools to gain insights in algorithmic behaviour. Several works now employ systematic experiments, qualitative comparisons between algorithms, or computational statistical techniques. Data-driven analyses have led to the development of algorithms that leverage the knowledge obtained on data collected on specific problems to make informed decisions during the search process. Automatic algorithm selection and configuration techniques are used to relieve practitioners of the burden of running extensive computational experiments and making unbiased decisions on the choice, respectively, of an algorithm and of a parameter configuration that are likely to perform well on an instance. However, they generate data about the effectiveness of each algorithm or parameter choice in the scenarios considered. This data can not only be used to drive the configuration process, but it can also be used to study algorithmic behaviour.

On the problem side, fitness landscape analysis aim to understand how the solution space of a certain problem instance impacts on the performance of a search algorithm. For example, it is easier for an algorithm to converge to a good solution in a smooth landscape, rather than on a more rugged one with lots of locally optimal solutions with poor quality.

In the literature, however, little work has been done to formally connect these related areas of research. The main issue lies in the difficulty of generalizing the many results obtained for specific problems, instances and algorithms to

other scenarios. In turn, this difficulty arises from the huge number of options we can choose from, on the stochasticity of the algorithms and instances, and on the computational burden of generating usable information, for example by computing features for a given problem. Algorithm selection usually relies on problem-specific instance features, so insights obtained for e.g. Traveling Salesman Problem (TSP) instances are not easily transferrable to SAT, QAP, or other combinatorial optimisation problems. Algorithm configuration can make an algorithm improve its performance, but current tools do not offer many explanations for their results, and common benchmarks are used mainly to raise the bar. The data generated can help in understanding algorithm performance, but to obtain insights the practitioner still needs to carefully inspect the outcome. Fitness landscape analyses are usually difficult to translate into an algorithm implementation or selection.

In this work, we propose a causal framework to represent the interaction between the entities involved in the resolution of an optimisation problem, and show how several approaches proposed to study algorithmic behaviour can be represented on this framework. With such models we can formally represent a theory or set of beliefs we hold about a certain system. A causal model [6] can be represented as a directed acyclic graph (DAG) whose nodes are the variables of the system under study. The variables can be divided into *exogenous* variables, whose values are determined by reasons external to the model, and *endogenous* variables, whose values are determined only by a subset of the other variables. The arcs encode the causal relationship between the variables, so that the value of an endogenous variable is probabilistically determined by the value of its parent nodes.

2 Causal models for SLS algorithms

Causal models provide an appropriate framework for this task, for several reasons. First, they explicitely encode the causality relationships between entities, and in our case such relationships are intuitively clear. In fact, we are convinced that in many works aimed at understanding algorithmic behaviour the authors already implicitely assume such causality relationships. On causal models we can perform inference, that is, estimate some unobserved variables given the value of other observed ones; and this is precisely how we consider the various tasks of algorithm selection and configuration, and several of approaches mentioned above. Causal models can also be used to analyse data collected from different sources and experiments.

While in this work we focus on trajectory-based SLS algorithms for non-constrained combinatorial optimisation problems, our framework can be extended to include other methods and different problem classes. We start from four working hypotheses, each one building on the previous one.

(H1) An algorithm can be divided into basic components and parameters. We take a component-based view of SLS algorithms [3,7], that is, we consider

an algorithm as a set of basic building blocks, each one possibly coming with a set of parameters, combined together in a certain way.

(H2) Separation between algorithm- and problem-specific components. Following (H1), algorithmic blocks are divided in problem-specific and algorithm-specific. Problem-specific components are the parts of a SLS that require specific knowledge of the problem under study, while algorithm-specific components are all the components that define what a SLS is and can be used across different unrelated problems. Thus, a SLS is defined by its algorithmic-specific components and their combination, and an instantiation of a SLS for a problem is the combination of the algorithm-specific components with a set of problem-specific components.

(H3) An algorithm operates a traversal of the search space. A SLS works by traversing solutions in a search space. The problem-specific components of a SLS are needed to (i) select one starting point of this traversal, and to (ii) evaluate another solution in the search space, relative to the current one. The separation of the problem-specific components of (H2) can be considered the simulation of an oracle that gives us the desired information about a solution when polled. Thus, we can ideally assume to have complete knowledge of the entire search space, turning any optimisation problem into a search problem for which we do not need additional knowledge on the problem anymore. This assumption lets us bridge the insights we obtain across different problems.

(H4) Identification of optimal algorithm behaviour. For obtaining optimal results on a search space, an algorithm needs to reach an optimal tradeoff and alternance of exploration and exploitation behaviour. This optimal behaviour differs for different landscapes, but several different algorithm can, in principle, reach this desired behaviour. In practice, we assume we can identify and configure an algorithm that obtains results that are "good enough" for our purpose.

Starting from these four working hypothesis, we build our causal framework. We consider problems and instances as given inputs to tackle, using an algorithm built starting from the basic components; the efficiency of the algorithm depends also on the computational environment (such as machine power or running time). These factors are the *causes* for the *effects*, the results obtained. What relates the inputs to the results, is how efficiently the algorithm can traverse the search space, that is, how it can efficiently balance diversification and intensification, based on the characteristics of the instance. These characteristics can be represented by the features and the fitness landscape.

The general causal framework representing the theory is shown in Figure 1. In the general framework each node is actually a macro-node representing a set of nodes, grouped together by their function.

High-level entities I represents the problem instances. P represents the problems. Following (H2), PA is the set of problem-specific components and algorithms (e.g. initial solutions, neighbourhoods, heuristics), and SA the set of

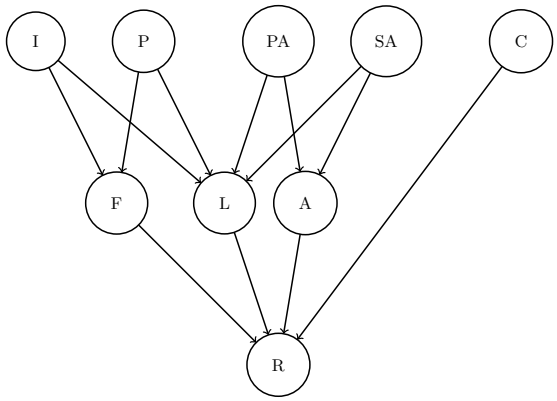

Fig. 1: A generic causal framework representing the interaction between problems, instances, algorithms and results. Arcs represent the causal relationships between the high-level nodes. In a practical application, not all the nodes might be observed, and not all the nodes might be present, depending on the specific instantiation of the problem under study.

algorithm-specific components that compose the search part of the algorithm. Components in PA and SA include both algorithmic functions such as neighbourhoods, acceptance criteria, ..., but also numerical parameters.

A is the set of algorithms that we can use for solving an instance $i_p \in I$ of problem $p \in P$; differently from PA and SA, an algorithm $a \in A$ is an instantiated and fully configured one. The separation of A from its components collected in PA and SA follows from (H1).

C represents the set of computational factors that impact an actual implementation of any algorithm $a \in A$, such as the computational power, running time, random seed, but also quality of the implementation, and any other possible factor affecting the computation.

F contains both the set of instance features, such as the instance size, and the problem-specific features. L represents instead the fitness landscape and the search space features, e.g. the ruggedness or the number of local optima.

Finally, R models the results that can be obtained by an algorithm $a \in A$ on an instance i_p for a problem $p \in P$, under the computational environment specified in C.

Causal relationships $\{I, P, PA, SA, C\}$ form the set of exogenous variables, the ones we set when we tackle an optimisation problem; $\{F, L, A, R\}$ are the endogenous variables, whose value depends on other variables, either deterministically or in a probabilistic way.

In F we include features defined solely on the instance itself and by both instance and problem, so we have incoming arcs from I and P. These features do not depend on the algorithm we choose, but are instead properties that arise when we define a specific problem instance.

In L, landscape features arise from the contribution of P and I; their differentiation from L is somewhat arbitrary, but by grouping them into a separate set we reflect in a clear way our working hypothesis (H3) and their "dynamic" characterization with respect to "static" features represented in F. In L we can include probing statistics or other features obtained with a phase of search space traversal by some search algorithm; for them we include arcs from PA and SA to L. Note, however, that the features generated using elements in PA are considered mere properties of the landscape, and not an evaluation of the algorithm (components). Thus, we do not include any arc to F or L from A, as it represents the algorithm used to produce the results, and not from C as we do not assume, in principle, a limitation on the possibility of computing features.

According to (H1) and (H2), an algorithm in A can be instantiated combining components from PA and SA, hence the incoming arcs.

Finally, the results R for an instance in I of a problem in P depend on the performance of an algorithm in A, under the computational environment represented in C; however, following our working hypothesis (H3), once an algorithm from A has been instantiated, it operates on a search space (here represented by L) and can be considered, at that point, unaware of the specific problem that generated F and L. In other words, F and L provide an intermediate (mediator) level of "reasons" that can help us in understanding the performance of A for I under the computational environment specified in C. The relationship here is probabilistic, for the stochastic nature of the algorithms and of the computational environment (hardware faults, etc.) that make effectively impossible to relate deterministically the causes with the results.

At this higher level, we do not need to assume any inter-node connection, that is, the nodes composing the high-level entities are not directly connected to each other, since they are defined either as exogenous variables, or as determined by other entities. Causal and inferential reasoning is as in any causal model [6]. Since there is no edge inside each macro-node, all the flows of information we define on the framework apply also when we "open" the macro-nodes into their constituting nodes.

3 Algorithm selection and configuration in the framework

The main goal of this work is to show how this causal framework can relate several existing approaches, in particular considering them as inference problems. For example, the Algorithm Selection Problem (ASP) requires to find the best algorithm in a portfolio of available ones (in our framework, $a \in A$) for a given set of instances (from I), which are characterized by a set of features (represented in F and L). With (H1) and (H2) we assume that we can instantiate at least one, and possibly more, algorithms from A. Since the selection is made by observing the results R for the alternatives evaluated, we can define ASP as the inference task of finding $a \in A$ given the observed variables in $\{R, F, L, C\}$.

Similarly, the Algorithm Configuration Problem (ACP) requires to find an optimal set of parameters and components (in SA and PA) for a given set of

instances (in I) of a problem in P, under computational constraints specified in C. The usual black box formulation of ACP does not make use of any feature, but makes the choice based only on the observed results R. Hence, the ACP can be formulated as the problem of inferring values for the variables in PA, SA given the observed variables in $\{P, I, R, C\}$.

Therefore, we can clearly see the relationship between ASP and ACP. They are both inference problems based on problem and instances, whose outcome depends on the results (in fact, the racing algorithm proposed in 1994 for model selection was subsequently used for configuring algorithms [1]). They instead differ in (i) whether we use fully configured algorithms (in A, for ASP) or building blocks (in PA and SA, for ACP), and (ii) if we can observe (and use in the selection or configuration process) some features in F or L. The separation is, however, neither crisp nor immutable. Some configurators can indeed make use of features [5]. By observing any feature in F or L in a configuration or PbO task we open the black box and move towards a selection setting; by allowing the configuration of numerical parameters of the algorithms in a portfolio for a selection task we have the CASH problem [8], instantiated as PA, PA given $\{R, F, L, C\}$.

We have defined a causal framework that models the causal relationships between the entities involved when tackling an optimisation problem. As first examples, we have shown how we can model ACP and ASP as inference tasks. However, several other approaches can be described according to this framework, thus contributing towards an improved ability of understanding and explaning how SLS algorithms work.

References

1. Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K.: A racing algorithm for configuring metaheuristics. GECCO 2002.
2. Cook, W.J.: Computing in combinatorial optimization. In: Steffen, B., Woeginger, G. (eds.) Computing and Software Science: State of the Art and Perspectives, LNCS, vol. 10000, pp. 27–47. Springer, Cham (2019).
3. Hoos, H.H.: Programming by optimization. Communications of the ACM **55**(2), 70–80 (Feb 2012).
4. Hoos, H.H., Stützle, T.: Stochastic Local Search—Foundations and Applications. Morgan Kaufmann Publishers, San Francisco, CA (2005)
5. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. LION 2011.
6. Pearl, J.: Causality: Models, Reasoning and Inference. Cambridge University Press, 2nd edn. (2009)
7. Stützle, T., López-Ibáñez, M.: Automated design of metaheuristic algorithms. In: Gendreau, M., Potvin, J.Y. (eds.) Handbook of Metaheuristics, Springer (2019).
8. Thornton, C., Hutter, F., Hoos, H.H., Leyton-Brown, K.: Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms. In: SIGKDD 2013.