Abstract

Summary: We introduce the caspo toolbox, a python package implementing a workflow for reasoning on logical networks families. Our software allows researchers to (i) learn a family of logical networks derived from a given topology and explaining the experimental response to various perturbations; (ii) classify all logical networks in a given family by their input-output behaviors; (iii) predict the response of the system to every possible perturbation based on the ensemble of predictions; (iv) design new experimental perturbations to discriminate among a family of logical networks; and (v) control a family of logical networks by finding all interventions strategies forcing a set of targets into a desired steady state.

Availability and Implementation: caspo is open-source software distributed under the GPLv3 license. Source code is publicly hosted at http://github.com/bioasp/caspo.

Contact: anne.siegel@irisa.fr
2 Overview

The workflow implemented by caspo is depicted in Figure 1. It starts with a PKN describing signed and directed signaling interactions and a dataset providing phosphorylation activities of a set of readouts with respect to several perturbations combining stimuli and inhibitions.

Learn. Given the PKN and the phosphoproteomics dataset, we aim at learning all logical networks compatible with the given topology and explaining the experimental observations similarly well (Guziolsowski et al., 2013; Videla et al., 2015b). Note that the number of logical networks in the family depends on the level of tolerance with respect to optimal fitness and size. In fact, this allows us to consider the uncertainty in observations and relax the parsimonious principle adopted for the learning.

Classify and predict. Once a family of logical networks is available, we are interested in classifying them with respect to their input–output predictions. Concretely, we look for sub-families of networks which cannot be distinguished based on the available experimental setup, i.e. the set of stimuli, inhibitions, and readouts. Note that logical networks within the same sub-family generate exactly the same output (readout) response for every possible input (stimuli and inhibitions), i.e., they are equivalent in terms of input-output. Thus, we refer to such sub-families of logical networks as logical input-output behaviors. Interestingly, the number of logical input-output behaviors is often significantly less than the number of logical networks (Guziolsowski et al., 2013) and this facilitates further analyses. For example, based on the input-output classification, we can compute the response of the system for every possible perturbation by combining the ensemble of predictions from all input-output behaviors. More precisely, for each possible input, the prediction for any output node will be the weighted average over the predictions from all input-output behaviors and where each weight corresponds to the number of networks exhibiting the corresponding behavior. Next, we can study the variability of each readout by means of the mean variance across all perturbations.

Design. As already noted, logical networks having the same input–output behavior cannot be discriminated based on the available experimental setup. However, alternative input-output behaviors could be distinguished by conducting further experiments. Thus, given a set of input-output behaviors, we are interested in designing new experimental perturbations which would allow for an optimal discrimination of rival models at hand (Videla et al., 2015a). Once the experiments are carried out in the wet lab, the new experimental observations would be combined with the previously available dataset and the workflow could start over.

Control. To conclude, given a family of logical networks we are interested in identifying key-players that would allow to control the response in every network. More precisely, we aim at finding minimal intervention sets that would force a set of targets into a desired steady state under various environmental conditions or constraints (Kaminski et al., 2013). For example, this could allow to find new therapeutic targets relevant for drug development or diagnosis. Consider two cell types, e.g. normal cells and cancer cells, and a family of logical networks describing the response in normal cells. Notably, such a family of networks will not reproduce the response of cancer cells to certain environmental condition. However, one could look for interventions that would force logical networks to reach the observed response in cancer cells. Then, such interventions could be interpreted as the mutations leading to cell dysfunction.

Each of the steps described above represents a challenging combinatorial optimization problem. In particular, our software strongly relies on answer set programming (ASP), a declarative modeling paradigm for which highly efficient solvers are available.

3 Implementation

Our software provides a command line interface (CLI) as well as an application programming interface (API) in order to facilitate the integration with other software packages and tools. Next, we provide a brief description of the CLI and refer the reader to the online documentation for an in-depth description of installation and usage (http://caspo.readthedocs.io). The CLI consists of various subcommands, viz., learn, classify, design, predict, control, visualize and test. Each subcommand provides its own help message describing required inputs and available options. Subcommands learn, classify, design, predict and control implement all the steps in the workflow described before and depicted in Figure 1. Each subcommand will output one or more files and some default visualizations. In general, the output of one subcommand corresponds to the input of another subcommand. This enables a straightforward application of the workflow for users without programming expertise. Finally, the subcommand test runs the complete workflow using various examples distributed with caspo.

Assumptions. We assume any signed and directed graph as a valid PKN typically describing signaling or regulatory events in the cell. However, while caspo accepts PKNs with loops, logical networks learned by caspo will not have loops since we focus on the response of the system at a single time-point reflecting a pseudo-steady state. It is worth noting that, an ASP-based method for learning logical networks with feedback-loops has been recently published (Ostrowski et al., in press) but is in general more limited in terms of the scale of systems it could possibly address. In this context, the classification of networks into input-output behaviors, the design of experiments in order to discriminate such behaviors, and the predictions made from such behaviors also assume logical networks without loops. In contrast, the control of logical networks is more general as it accepts any logical network (with or without loops). In caspo learn, classify, design and predict we assume an experimental setup fixed, i.e. a fixed set of possible stimuli, inhibitors and readouts. Further, in caspo learn we assume a dataset describing the response of the system to multiple experimental perturbations...
Table 1. Description of three case studies, i.e. PKN and dataset

| Scale | Nodes | Edges | Hyperedges | Perturbations | Readouts | MSE | Size | Networks | \(t_{opt}\) | \(t_{enum}\) | I/O | ts | Designs | Perturbations/ | Design | \(t_{opt}\) | \(t_{enum}\) |
|-------|-------|-------|------------|--------------|----------|-----|------|----------|---------|---------|------|----|---------|----------|--------|---------|---------|
| S     | 17    | 32    | 77         | 25           | 175      | 15  | 0.0395 | 178      | 0.06    | 0.28    | 5    | 3   | 1       |         | 0.01   | 0.02   |
| M     | 31    | 53    | 130        | 64           | 960      | 28  | 0.0499 | 144      | 0.24    | 0.57    | 4    | 2   | 2       |         | 0.01   | 0.02   |
| L     | 45    | 110   | 265        | 120          | 1920     | 52  | 0.1317 | 384      | 516.73  | 174.11  | 10.59| 9   | 3       | 0.02    | 0.04   |
| XL    | 45    | 110   | 489        | 120          | 1920     | 52  | 0.1317 | 384      | 1501.81 | 3367.42 | 10.59| 9   | 3       | 0.02    | 0.04   |

Both, L and XL correspond to the same PKN and dataset but L is limited to hyperedges with up to 2 source nodes (which yields logical networks having AND gates with up to 2 inputs) while XL considers any possible hyperedge. In learn we show optimum mean squared error (MSE) and size, number of networks within certain MSE and size tolerance (10% and 2 for S, 2% and 0 for M, and no tolerance for L and XL), computation time for finding the optimum \(t_{opt}\) and for enumeration of all optimal networks \(t_{enum}\). In classify we show the number of input-output behaviors and the computation time \(t_{as}\). Finally, in design we show the number of optimal experimental designs, the number of experimental perturbations per design, and computation time for finding the optimum \(t_{opt}\) and for enumeration of all optimal designs \(t_{enum}\). All computation times shown are reported in seconds.

Combining stimuli and inhibitors. Such a system response must be provided in terms of measurements over the set of readouts at a single time-point and normalized in the range \([0,1]\). Although the methods implemented in caspo were initially developed for signaling networks and phosphoproteomics datasets, any PKN and dataset satisfying these assumptions could also work.

Case studies. We consider three real-world and recently published case studies modeling signaling pathways of primary human hepatocytes in liver cells. In Table 1 we describe the main characteristics for each PKN and dataset. Apart from number of nodes and edges for each PKN, we also show the number of possible hyperedges \(b\) derived from the PKN. Such a number is particularly relevant for learning logical networks since the number of possible networks to explore is given by \(2^b\) (Saez-Rodriguez et al., 2009). The small-scale case study (S) consists of a PKN and a dataset from the DREAM4 challenge (Prill et al., 2011). The medium-scale case study (M) consists of a PKN and a dataset introduced in Saez-Rodriguez et al. (2009). Finally, the large-scale case study (L) consists of a PKN and a dataset published by Melas et al. (2012). Further, in this case we consider either to limit the number of sources per hyperedge to 2 (up to 2 inputs per AND gate) or not (XL). Notably, this has a direct impact in the search space of logical networks to explore during learning.

Performance. We illustrate the performance of caspo over each case study for learn, classify, and design. For control, we refer the reader to Kaminski et al. (2013). Benchmarks shown in Table 1 were run using a dedicated server with 16 cores and 60G of RAM. Notably, running caspo using 16 threads allows us to explore the search space using 16 different search heuristics in parallel. The learning of logical networks for the small and medium-scale case studies is relatively easy for caspo while the large and extra large-scale case studies increase computation time in three and four orders of magnitude, respectively. For the large and extra large-scale case studies (with and without limiting the number of inputs per AND gate), caspo finds the same family of 384 optimal logical networks. Thus, for the following analyses in the workflow (classify and design) L and XL are reported as one case study. As shown in Table 1 several hundred of networks can be classified by caspo into the corresponding input-output behaviors in a few seconds. However, classification of a logical networks family into input-output behaviors is highly dependent in the number of networks in such a family. For example, if we increase the tolerance over optimum MSE for the medium-scale case study to 8%, caspo finds 3524 logical networks in 8 seconds. In that case, such a family of networks is classified into 66 input-output behaviors in \(\sim\)10 min. Similarly, finding all optimal experimental designs in order to discriminate among a few input-output behaviors can be very fast as shown in Table 1. Nonetheless, as the number of input-output behaviors increases, the computation time required to find all optimal experimental designs could also increase significantly. For example, for the family of 66 input-output behaviors, caspo finds 6 optimal experimental designs made of 7 perturbations each in \(\sim\)3 h.

4 Conclusion

The caspo toolbox provides a logic-based implementation of the hypothesis-driven research loop in systems biology. Combining various steps, viz., learn, classify, design, predict and control, it enables a complete study of the variability of logical networks families from various perspectives.

Our software could be extended in several ways. In particular, the learning of logical networks could allow users to specify a set of fixed logic rules and learn the rest. Also, feasible interactions are likely to be missing from any given PKN. Thus, the ability to infer new links that would improve the fitness to a given dataset could be very useful. Moreover, links in the PKN could be assigned to different levels of confidence depending on the methods used to build such a PKN and weighted accordingly in the objective function (Eduati et al., 2012). Regarding performance, while caspo is able to cope with real-world case studies very efficiently, various factors should be taken into account, especially for learning logical networks. Of course, the size of the PKN is determinant but also the amount and type (single versus combinatorial) of experimental perturbations is critical in order to constrain the search space. Further, tolerances with respect to optimum MSE and size could lead to an intractable number of logical networks if they are not chosen carefully and rather conservatively. Altogether, the caspo toolbox is a powerful software that we expect to keep improving as more researchers start using it for their investigations.

Funding

Biotempo ANR-10-BLANC-0218 and TGFSysBio (ITMO Cancer, Plan Cancer 2014–2018).

Conflict of Interest: none declared.
References

Abou-Jaoude, W. et al. (2016) Logical modeling and dynamical analysis of cellular networks. Front. Genet., 7, 94.
Eduati, F. et al. (2012) Integrating literature-constrained and data-driven inference of signalling networks. Bioinformatics, 28, 2311–2317.
Guziolowski, C. et al. (2013) Exhaustively characterizing feasible logic models of a signaling network using answer set programming. Bioinformatics, 29, 2320–2326.
Kaminski, R. et al. (2013) Minimal intervention strategies in logical signaling networks with ASP. Theory Pract. Logic Program., 13, 675–690.
Melas, I.N. et al. (2012) Construction of large signaling pathways using an adaptive perturbation approach with phosphoproteomic data. Mol. BioSyst., 8, 1571–1584.
Ostrowski, M. et al. Boolean network identification from perturbation time series data combining dynamics abstraction and logic programming. Biosystems, (in press).
Prill, R.J. et al. (2011) Crowdsourcing network inference: The dream predictive signaling network challenge. Sci. Signal., 4, mr7.
Saez-Rodriguez, J. et al. (2009) Discrete logic modelling as a means to link protein signalling networks with functional analysis of mammalian signal transduction. Mol. Syst. Biol., 5, 331.
Videla, S. et al. (2015a) Designing experiments to discriminate families of logic models. Front. Bioeng. Biotechnol., 3, 131.
Videla, S. et al. (2015b) Learning boolean logic models of signaling networks with ASP. Theor. Comput. Sci., 599, 79–101.
Wang, R.S.R. et al. (2012) Boolean modeling in systems biology: an overview of methodology and applications. Phys. Biol., 9, 055001n model.