Computing Diverse Boolean Networks from Phosphoproteomic Time Series Data

Misbah Razzaq[1], Roland Kaminski[2], Javier Romero[2], Torsten Schaub[2], Jeremie Bourdon[3], and Carito Guziolowski[1(✉)]

[1] Laboratoire des Sciences du Numerique de Nantes, Ecole Centrale de Nantes, Nantes, France
{misbah.razzaq,carito.guziolowski}@ls2n.fr
[2] University of Potsdam, Potsdam, Germany
{kaminski,javier,torsten}@cs.uni-potsdam.de
[3] Laboratoire des Sciences du Numerique de Nantes, Universite de Nantes, Nantes, France
jeremie.bourdon@ls2n.fr

Abstract. Logical modeling has been widely used to understand and expand the knowledge about protein interactions among different pathways. Realizing this, the *caspo-ts* system has been proposed recently to learn logical models from time series data. It uses Answer Set Programming to enumerate Boolean Networks (BNs) given prior knowledge networks and phosphoproteomic time series data. In the resulting sequence of solutions, similar BNs are typically clustered together. This can be problematic for large scale problems where we cannot explore the whole solution space in reasonable time. Our approach extends the *caspo-ts* system to cope with the important use case of finding diverse solutions of a problem with a large number of solutions. We first present the algorithm for finding diverse solutions and then we demonstrate the results of the proposed approach on two different benchmark scenarios in systems biology: (1) an artificial dataset to model *TCR signaling* and (2) the *HPN-DREAM* challenge dataset to model breast cancer cell lines.

Keywords: Diverse solution enumeration · Answer set programming Boolean Networks · Model checking · Time series data

1 Introduction

Network analysis methods have been widely used for studying phosphoproteomic data, yielding important insights into protein interactions, functions, and evolution. Several formalisms including differential equations, Boolean logic and fuzzy logic exist for modeling signaling networks [4,26,28]. Models elucidated using differential equations require explicit specifications of kinetic parameters of the system and work well for smaller systems. Despite being highly predictive, mathematical modeling becomes computationally intensive as networks become

M. Češka and D. Šafránek (Eds.): CMSB 2018, LNBI 11095, pp. 59–74, 2018.
https://doi.org/10.1007/978-3-319-99429-1_4

larger. Stochastic modeling is suitable for problems of random nature but also fails to scale well with large scale systems of proteins [16,28].

On the other hand, Boolean network (BN) modeling [14] has demonstrated to be a powerful framework for studying signaling networks [1] and for predicting novel behavior under perturbations. Phosphoproteomic data shows alteration in protein levels under different perturbation. Several methods have been proposed for learning BNs from such data. Most of the methods restrict their focus on one time point only [9,17,23,27], which prevents them from capturing interesting dynamic characteristics such as loops [16]. Realizing this, methods have been proposed to model time series data [5,20,24]. Given noisy experimental data, most existing methods based on integer linear programming [17] and answer set programming (ASP) [19,27] infer a family of BNs, which equally well represent the underlying signaling behavior in different pathways.

In this study, we focus on the ASP-based *caspo-ts* system which learns a family of BNs from time series data and a prior knowledge network (PKN). *caspo-ts* uses an over-approximation to learn candidate BNs, which leads to some false positive (FP) BNs. These BNs are not guaranteed to reproduce all traces of the time series data. To resolve this issue, it uses exact model checking to filter out FP BNs. The *caspo-ts* method uses the *clingo* ASP solver [7], which is able to exhaustively enumerate all solutions. The *clingo* solver by default uses an enumeration scheme, in which, once a solution is found, it backtracks to the first point from where the next solution can be found. This typically leads to the situation where successive solutions only change in a small part. As a result, *caspo-ts* may enter a solution space where FP BNs are clustered together. Given the size of the PKN and the small number of perturbations in the experimental data, the solution space of the *caspo-ts* can be very large containing billions of BNs making it difficult to enumerate true positive (TP) BNs in reasonable time if it gets stuck in a cluster of FP BNs.

To overcome this, we extend *caspo-ts* with a new enumeration scheme for breaking up clusters of similar solutions. In [6], various methods were presented for computing diverse solutions in ASP. However, these methods are not applicable to *caspo-ts*, since this system enumerates optimal (subset minimal) solutions, in order to produce simpler and more relevant solutions. Instead, we extend the approach of [21] for computing optimal diverse solutions[1] in ASP. The novelty of this extension is that we use heuristics for both the computation of optimal (subset minimal) solutions, and the diversification. By sampling the large solution space of BNs, we can retrieve a more complete set of mechanisms explaining the experimental data and better approximate biological reality.

Regarding model refinement of BNs dynamics using solvers, the works of [2,22] propose ways to discover BNs or prune them according to experimental data related to fix points or attractors; which represent key biological functions. The objective is to find mechanisms explaining these biological functions. Their

[1] In the following, a diverse optimal solution is a solution which is minimal w.r.t. an objective function, there is no solution which is a subset of it, and it is different from previously enumerated solutions.

results are exhaustive, notably focusing on multiple mechanisms. Compared to [2,22], we propose a method that handles large scale networks and time-series phosphoproteomic data. The advantage of this is that such data is derived from standard experimental protocols. Moreover, the networks we handle are inferred from publicly available databases. Our method also allows us to provide optimal data, with respect to noisy or incomplete datasets. In this sense our method adapts more to current high-throughput experimental technologies as well as to massive signaling knowledge sources.

In the following, we refer to the modified *caspo-ts* as *caspo-ts*^{D}. We apply both systems to two datasets: (1) an artificial dataset for network signaling model, and (2) the HPN-DREAM challenge dataset. Our results show substantial improvements of *caspo-ts*^{D} in solution quality by discovering more signaling behaviors than *caspo-ts*. Moreover, *caspo-ts*^{D} is able to find solutions in cases where *caspo-ts* is unable to find any. Our method is applicable to gene or protein expression time series datasets measured upon different perturbations. Moreover, the proposed method is not specific to our biological application. It computes diverse subset minimal solutions in ASP, and therefore can be applied to any problem modeled in ASP.

The remainder of the paper is structured as follows. In Sect. 2, we describe the datasets and the algorithms. In Sect. 3, we study the performance of the modified enumeration scheme on the artificial and real datasets. In Sect. 4, we give concluding remarks and describe future work.

2 Materials and Methods

In this subsection, we describe the datasets, the *caspo-ts* system, and the new algorithm to enumerate diverse BNs implemented in *caspo-ts*^{D}.

2.1 Phosphoproteomic Time Series Dataset

Here, we give a brief description of the phosphoproteomic datasets used for testing the performance of the extended *caspo-ts*^{D} system. Phosphoproteomic data show changes in protein levels under sets of perturbations. Here, proteins are referred to by three names: (1) stimuli, (2) inhibitors, and (3) readouts. Stimuli serve as interaction points for the experimentalist. Inhibitors are blocked over all time points of the perturbation. Readout proteins are measured under sets of perturbations at different time points. Perturbations are a combination of stimuli and inhibitors. Fig. 1 depicts an example of phosphoproteomic time series data, where the values between zero and one of three proteins are shown in different colors. In this figure, we see the time series of one readout protein (blue) under a perturbation of one stimulus (green) and one inhibitor (red). Stimuli have value 1 and inhibitors have value 0 across all time points of an experimental perturbation. Readouts take continuous values in [0;1] after normalization. In some phosphoproteomic datasets an inhibitor can also act as a readout protein, which means that there are perturbations where it will be measured.

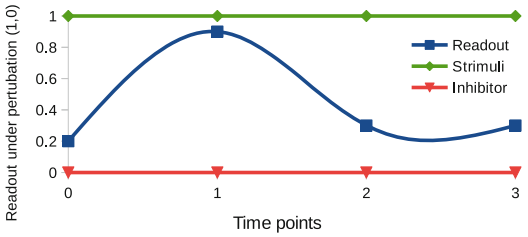

Fig. 1. Phosphoproteomic time series data. (Color figure online)

Artificial Dataset. The artificial dataset for TCR signaling was generated by [19] by simulating the PKN using logic based ODEs. This dataset consists of 4 readouts, 3 stimuli and 2 inhibitors. The readout proteins were measured at 16 time points under 10 perturbations. The PKN was derived from the TCR signaling model of [15] and consists of 16 nodes and 25 edges.

HPN-DREAM Dataset. The HPN-DREAM dataset consists of phosphoproteomic data of four breast cancer cell lines (BT20, UACC812, MCF7, BT549). This dataset was downloaded from the web portal of the HPN-DREAM challenge [11,12]. It includes temporal changes in phosphorylated proteins at seven different time points ($t_1 = 0\,\text{min}$, $t_2 = 5\,\text{min}$, $t_3 = 15\,\text{min}$, $t_4 = 30\,\text{min}$, $t_5 = 60\,\text{min}$, $t_6 = 120\,\text{min}$, and $t_7 = 240\,\text{min}$) under sets of perturbations. Maximum value based normalization was applied to the data to bring values into the range $[0; 1]$ and noisy and incomplete time series data was removed. After this, we have approximately 23 phosphorylated readout proteins per cell line. The number of perturbations varies from one cell line to another. The main goal of the HPN-DREAM challenge is to learn context specific signaling networks efficiently and effectively to predict dynamics in breast cancer.

The PKN was generated by mapping the experimentally measured phosphorylated proteins (HPN-DREAM dataset) to their equivalents from literature-curated databases and connecting them together within one network. The PKN was built using the ReactomeFIViz (Cytoscape app), which accesses the interactions existing in the Reactome and other databases [29]. The PKN consists of 64 nodes (7 stimuli, 3 inhibitors, and 23 readouts) and 178 edges.

2.2 Caspo-ts

The *caspo-ts* system is based on a combination of ASP and model checking. The ASP part of the *caspo-ts* system is used to solve the combinatorial optimization problem of finding BNs compatible with a PKN and time series data. All learned BNs are optimized using an objective function, minimizing the distance between the original and the time series data determined by the BN learned with *caspo-ts*. The ASP solver guarantees finding all optimal solutions w.r.t. an objective function. The model checking part of the system detects TP BNs by checking

the reachability of time series traces given compatible BNs generated by the ASP part of the system. TP BNs are guaranteed to reproduce all the (binarized) traces under all perturbations by verifying reachability in the BN state graph. Since checking this reachability is a PSPACE-hard problem, the second step can be very time consuming for large BNs. To resolve this issue, the ASP part over-approximates solutions [19]. This over-approximation removes a large set of BNs that have no reachable traces, reducing the number of calls to the model checker.

The BNs learned by *caspo-ts* are represented by Boolean formulas in Disjunctive Normal From (DNF), i.e., as a disjunction of conjunctive clauses [2]. The BNs inferred by *caspo-ts* use the smallest DNF formulas possible, in the sense that no conjunctive clause can be removed from a DNF formula without changing the Boolean function it represents. We refer to these BNs as subset minimal BNs.

In the following, we give a brief description of ASP and the solving algorithms used by *caspo-ts*. But first let us have a look at an example Boolean formula.

Example 1. To use Boolean formulas we discretize the phosphoproteomic data: values greater or equal to .5 are set to 1, and to 0 otherwise. Let protein A have the Boolean formula $(B) \vee (\neg B \wedge C)$ containing the two conjunctive clauses (B) and $(\neg B \wedge C)$, where B and C represent proteins. This formula can be used to update the value of protein A. If the update is applied, A is set to 1 if either the value of B is 1, or the value of B is 0 and the value of C is 1. Otherwise, the value of A is set to 0.

Answer Set Programming. A *logic program* consists of *rules* of the from

$$h \leftarrow b_1 \wedge \cdots \wedge b_m \wedge \neg b_{m+1} \wedge \cdots \wedge \neg b_n$$

where h is an atom, $0 \leq m \leq n$, and each b_i is an atom. Such a logic program induces a set of stable models determined by the stable model semantics; see [8] for details. Each stable model is a subset of the atoms occurring in the logic program. Atoms appearing in this set are said to be true, and false otherwise. A rule is satisfied if its body (the part after the $\leftarrow$) is not satisfied, or its head atom h is true. A rule body is satisfied if all the atoms b_1 to b_m are true and all the atoms b_{m+1} to b_n are false. A *stable model* satisfies all rules of a logic program and also satisfies a minimality criterion. We do not go into the full details here, but this criterion requires that each atom in a stable model is proved by some rule. For this, a true atom has to appear in at least one rule head with a satisfied body. In the following, we simply refer to the stable models of a logic program as its solutions.

We use two extensions [25] to logic programs, which are frequently used in practice and ease modeling problems with ASP. A *choice rule* has form

$$\{h_1, \ldots, h_o\} \leftarrow b_1 \wedge \cdots \wedge b_m \wedge \neg b_{m+1} \wedge \cdots \wedge \neg b_n$$

[2] A clause can be seen as a reaction, where the proteins represented positively are available, and the proteins represented negatively are absent. A Boolean formula in DNF encompasses all possible reactions to update the value of a protein.

where $1 \leq o$ and each h_i is an atom. Unlike with the normal rule above, a choice rule can be used to prove any subset of the atoms h_1 to h_o whenever its body is satisfied. A *constraint* has form

$$\leftarrow b_1 \wedge \cdots \wedge b_m \wedge \neg b_{m+1} \wedge \cdots \wedge \neg b_n$$

and it removes all solution candidates that satisfy its body, without proving any atoms.

Example 2. Using a single choice rule, the solutions of the program

$$\{a, b, c\} \leftarrow \tag{1}$$

are all the subsets of the set $\{a, b, c\}$. To build up our running example, we further add the following constraints:

$$\leftarrow b \wedge \neg a \wedge \neg c \tag{2}$$

$$\leftarrow \neg b \wedge c \tag{3}$$

$$\leftarrow \neg b \wedge \neg c \tag{4}$$

The first constraint discards all solutions where b is true, a is false, and c is false. The second those where b is false and c is true, and the third those where both b and c are false. Hence, for the above program, we obtain the solutions $\{a, b\}$, $\{b, c\}$, and $\{a, b, c\}$.

ASP Solving. Next, we describe how the ASP solver *clingo* used by *caspo-ts* discovers solutions (BNs) using the conflict driven clause learning algorithm [7], shown in Algorithm 1.

```
   Input: program P
1  Initialize assignment;
2  while assignment is partial do
3  |  Decide;
4  |  Propagate;
5  |  if propagation let to a conflict then
6  |  |  Analyze;
7  |  |  if conflict can be resolved then
8  |  |  |  Backjump;
9  |  |  else
10 |  |  |  return unsatisfiable;
11 return solution given by assignment;
```

Algorithm 1: Conflict-driven clause learning.

The algorithm works by extending a Boolean assignment over the atoms occurring in the given logic program P until a solution is found. The assignment is initialized in line 1. Then it is extended by the decision heuristic and

propagation in the loop in lines 2–10. The call to `Decide()` in line 3 at the beginning of the loop uses a heuristic to select an atom, makes it either true or false, and adds it to the assignment. The consequences of this decision are then propagated in the following line extending the assignment accordingly. Then it is checked if propagation leads to a conflict. If this is the case, then the conflict is analyzed in line 6 and the assignment adjusted in line 8 accordingly. Note that a call to `Backjump()` takes back one or more decisions together with their consequences, and then adds an additional consequence to the assignment. This property ensures that the algorithm always terminates. It can also happen that a conflict cannot be recovered from. In this case, the problem is found unsatisfiable and the algorithm returns in line 10. Once the assignment is complete, the corresponding solution (set of true atoms) is returned in line 11.

Example 3. We can now apply this algorithm to our running example (c.f. Example 2). Starting with an empty assignment, we set a to false as the first decision. There are no immediate consequences and, hence, no conflict can arise. Then we decide to make b false. The consequences of this decision are that c is false via rule (3), and c is true via rule (4). Hence, we get a conflict, which is resolved and followed by a backjump. Since the conflict was caused by deciding a truth value for b, but is independent of the decision for a, the algorithm takes back all decisions and adds b as a consequence (we now know it must be true in all solutions). We can then decide to make a false again, which sets c to true via rule (2). This decision does not cause a conflict and the assignment is no longer partial. Hence, the algorithm terminates with solution $\{b, c\}$.

2.3 Caspo-ts^D

Here, we describe the algorithm used for enumerating diverse subset minimal solutions. In *caspo-ts*, the algorithm is implemented in the Python programming language using *clingo*'s multi-shot solving API [13]. The API allows us to customize the solving process, in particular, it allows us to customize the decision heuristic of the solving component, which is the key feature to find subset minimal answer sets. Note that we implemented the algorithm using the multi-shot solving API of *clingo* version 5. For that, we have upgraded the solver *clingo* of *caspo-ts* from 4.5.4 to 5.

Algorithm 2 is used to enumerate subset minimal answer sets. The idea is to configure the decision heuristic in lines 8 and 10 (see function `Decide()` in Algorithm 1 line 3) so that it first makes all atoms subject to subset minimization false before deciding truth values for other atoms. This modification ensures that the solution obtained from Algorithm 1 by calling `Solve()` is a subset minimal solution (see [3] for more details). Such a solution is output in line 12 of the algorithm. Furthermore, the algorithm calls `Solve()` in line 5 multiple times to find *all* subset minimal solutions. To not enumerate solutions twice, a constraint preventing to find the same solution again is added to the logic program P in the following line. This constraint is violated whenever a superset of the atoms

```
   Input: program P and atoms T to subset minimize
 1 Prepare(P);
 2 foreach x ∈ T do
 3     SetSign(x, false, 1)
 4 while satisfiable do
 5     S ← Solve();
 6     AddConstraint(← a_0, ..., a_n for {a_0, ..., a_n} = T ∩ S);
 7     foreach x ∈ T ∩ S do
 8         SetSign(x, false, 2);
 9     foreach x ∈ T \ S do
10         SetSign(x, false, 1);
11     if S is a true positive then
12         Output(S);
```

Algorithm 2: Diverse subset minimal solution enumeration.

in the previously found solution is true. This process is repeated in the loop in lines 4–12 until the program is no longer satisfiable and, hence, all solutions have been enumerated.

So far we only discussed how to enumerate subset minimal solutions. Now we explain how to extend the method in order to compute diverse solutions. The key idea to make the next solution different from the previous one is to assign atoms appearing in the last solution to false before assigning any other atoms. To modify the heuristic, we use function `SetSign`(a, t, l), which instructs the decision heuristic to assign atom a to truth value t on level l. The decision heuristic assigns free atoms with the highest level to the designated truth values before assigning atoms on lower levels. By default all atoms have level 0 and the decision heuristic is free to make them either *true* or *false*. The loop in lines 7–8 instructs the decision heuristic to assign atoms that appeared in the last solution to false on level 2. Any other atoms subject to subset minimization are assigned to false on level 1 in lines 9–10. And since all other atoms by default have level 0, they are assigned last. We see in the experiments in the next section that this strategy breaks up clusters of similar solutions in the solution sequence.

Example 4. We continue with Example 2. Let us assume that a, b, and c are the atoms subject to subset minimization. Note that in Example 3 all decisions assigned atoms to false, so the first solution $\{b, c\}$ obtained is in fact a subset minimal solution. Let us further assume that Algorithm 2 produced this solution in the first iteration (line 5). First, the constraint $\leftarrow b \wedge c$ preventing any superset of $\{b, c\}$ as solution is added in line 6. Then, the decision heuristic is configured to set atoms b and c to false on level 2 (line 8) and atom a to false on level 1 (line 10). In the next iteration, the only possible decision is to set c to false because b is already irrevocably assigned. The consequence of this decision is to set a to true via rule (2). Hence, we obtain solution $\{a, b\}$, which is a subset minimal solution. This is followed by adding the constraint $\leftarrow a \wedge b$ in line 6,

which in turn makes the program unsatisfiable and causes the loop in lines 4–12 to terminate. We correctly obtain the subset minimal solutions $\{b, c\}$ and $\{a, b\}$. Solution $\{a, b, c\}$ is not subset minimal and not enumerated.

3 Results

In this section, we discuss the results of applying Algorithm 2 on two different datasets. We start with the artificial benchmark, where the solution space is small enough to compute all solutions. This allows us to study this benchmark in more detail. We can analyze how well a limited number of solutions enumerated with *caspo-ts* and *caspo-ts*^{D} represents the solution space. Then we move to the real dataset, where we cannot enumerate all solutions and can only consider a limited number of solutions because the solution space is too large. Nevertheless, we can show improved results with *caspo-ts*^{D} over *caspo-ts* by being able to enumerate more TP BNs and also more diverse BNs.

3.1 Artificial Dataset

Here, we use the TCR signaling dataset [15] to demonstrate the working of Algorithm 2 by describing two factors: (1) frequency of the clauses, and (2) true positive rate of BNs. The purpose of studying the first factor is to observe how many clauses we discover while learning a limited number of BNs. In this case, we expect to discover more clauses with *caspo-ts*^{D} than with *caspo-ts*. The second factor (the true positive rate) is used to study how TP solutions are distributed in the solution space. With *caspo-ts*^{D}, we expect TP solutions to be distributed much more evenly.

Figure 2 depicts the frequency of clauses in the solutions of the artificial benchmark. Clauses that occurred in at least one solution are depicted on the x axis. Each tick stands for one clause and the label has format $n \leftarrow c$ where c is a clause and n is a node name. Furthermore, clauses associated with the same node are grouped together by shading the background alternatingly in light gray and white. The frequencies of the clauses are depicted on the y axis. The red line depicts the frequency considering all 68338 solutions, while the green and blue lines depict the frequencies of the first 100 solutions computed by *caspo-ts* and *caspo-ts*^{D}, respectively. In total, there are 49 clauses appearing in the family of BNs. While *caspo-ts*^{D} (blue line) discovered 48 clauses, *caspo-ts* (green line) learned only 29 clauses by enumerating the same number of BNs (100). We also observe that the blue line is often much closer to the red line (with an average distance of 0.06) than the green line is (with an average distance of 0.20). This shows that the diverse enumeration scheme is able to produce solutions that are less similar to each other and better represent the solution space. The underlying ASP solver of *caspo-ts* by default uses an enumeration scheme that backtracks to the first point from which the next solution can be found. This approach typically

leads to the situation where successively enumerated solutions only change in a small part. We can observe this behavior in Fig. 2, where some clauses are overrepresented.

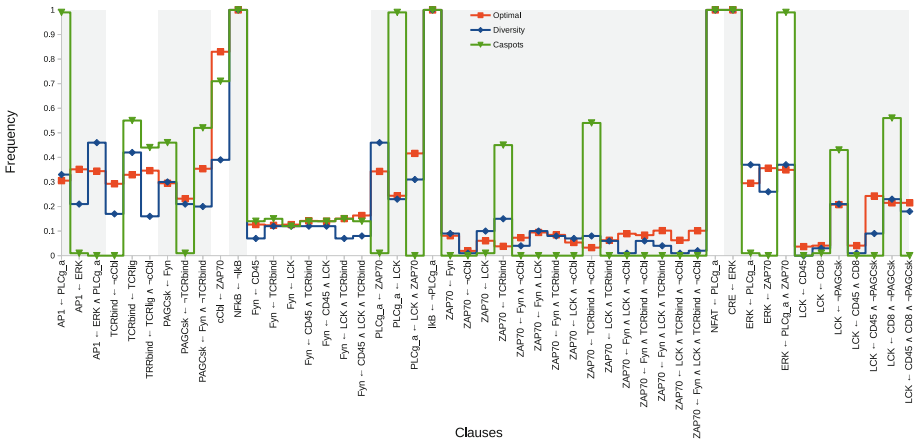

Fig. 2. Frequency of clauses per node in all 68338 BNs (red line), and in the first 100 BNs enumerated by *caspo-ts* (green line) and *caspo-ts*^{D} (blue line). (Color figure online)

Figure 3 depicts the true positive rate of blocks of successive solutions. Each tick on the x axis stands for a block of 1000 solutions. The y axis depicts the percentage of true positives in a block of solutions. The red line depicts the overall true positive rate (78%), while the green and blue lines depict the true positive rates of *caspo-ts* and *caspo-ts*^{D}, respectively.[3] We observe that for the *caspo-ts* system there are a lot of blocks with either a lot of true positives or very few. This suggests that true positives are clustered in the sequence of enumerated solutions. We observe that the diverse enumeration scheme does not show this behavior. This is especially important for enumerating true positive solutions of real world instances where only a limited number of solutions can be checked because of time constraints. With the original *caspo-ts* system, it can happen that the first cluster does not contain any true positives, making it impossible to find any true positive solution within a given time budget. The graph also shows that the diverse enumeration scheme does not sample over the full solution space. We see that before around 23000 solutions, the true positive rate is below the ideal 78% and then jumps up afterward. Thus, we conjecture that our enumeration scheme mainly breaks up local clusters of solutions with the artificial benchmark. Still, it is able to discover almost all clauses compared to the whole solution set (the frequency is 0 only once), while *caspo-ts* does not discover 20 clauses at all.

[3] For example, when x has value 3000, the y value in blue gives the true positive rate among the solutions 2001 to 3000 computed by *caspo-ts*^{D}.

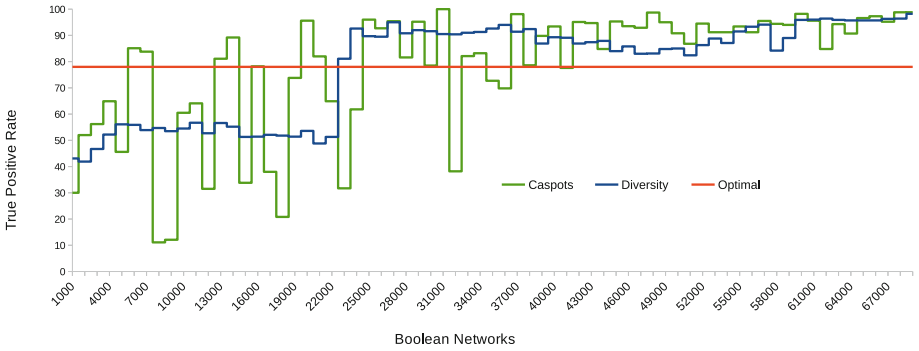

Fig. 3. True positive rate of BNs grouped in blocks of 1000 networks. (Color figure online)

3.2 HPN-DREAM Challenge Dataset

Next, we show the results of applying diverse solution enumeration to the HPN-DREAM challenge dataset [11]. We discuss the results according to three aspects: (1) time to compute the first true positive BN, (2) similarity among the family of solutions, and (3) Boolean functions computed by the original *caspo-ts* and the extended *caspo-ts*^{D} system. We start the analysis with four cell lines, and then we provide a detailed analysis of the Boolean functions of one cell line discovered by *caspo-ts* and *caspo-ts*^{D}.

Given that model checking is a computationally hard problem, we stop an experiment after a system verifies (using the model checker) 46 BNs per cell line[4]. The model checking task was performed on a server with 1.5 Tb of RAM. Table 1 shows the number of TP BNs obtained for each cell line. We see that the number of TP BNs differs comparing *caspo-ts* and *caspo-ts*^{D}. For MCF7 we obtain 0 TP BNs with *caspo-ts* and 4 TP BNs with *caspo-ts*^{D}, while for BT549 we obtain 2 and 14 TP BNs, respectively. For the other two cell lines BT20 and UACC812, we obtain a comparable number of BNs. We observe that we can get more TP solutions by checking the same number of BNs with *caspo-ts*^{D}. This is an important improvement given the fact that model checking the BNs is a computationally hard problem. Next, we consider the time column showing the time to compute the first TP BN for each cell line. We see that we are unable to get TP BN with *caspo-ts* in case of the MCF7 cell line, which shows that the *caspo-ts* system is stuck in a part of the search space where there are only FPs. Otherwise, for the other cell lines the time to get the first TP BN is comparable. We conclude that the difficulty to model check a BN depends on the cell line and not on the order in which solutions are found. Finally, the similarity column shows the similarity score among the set of TP BNs for each

[4] Note that the model checker could only verify 32 out of 46 solutions within one month for cell line BT20 in case of *caspo-ts*^{D}. There may exist more TPs for this cell line.

cell line. This score is calculated by comparing the clauses of one cell line with each other. We observe that the similarity among the solutions is much higher for *caspo-ts* than for *caspo-ts*^{D}. From this we conclude that, studying the same number of networks, *caspo-ts*^{D} can discover more clauses representing diverse signaling behaviors.

Table 1. Number of TP BNs out of 46 BNs, time to get the first TP solution, and similarity among TP solutions per cell line.

	caspo-ts			*caspo-ts*^{D}		
Cell Line	TPs	Time	Similarity	TPs	Time	Similarity
MCF7	0	—	—	4	6.7 h	0.51
BT549	2	8.4 min	0.92	14	7.9 min	0.44
UACC812	20	26 s	0.81	15	27 s	0.45
BT20	13	20 h	0.86	7+	20 h	0.32

Now, we analyze in more detail the UACC812 cell line using *caspo-ts*. Fig. 4 shows the union of 10 TP BNs obtained by *caspo-ts*. There are four different kinds of nodes in the graph: (1) stimuli shown in green, (2) inhibitors shown in red, (3) readouts shown in blue, and (4) unobserved nodes shown in white. Note that blue nodes with red borders are readouts, which are also inhibitors. There are two different kinds of edges shown in red and green color. Green edges are used to show a positive influence ($\leftarrow$), and red edges are used to show a negative influence ($\vdash$). We have discovered 25 clauses with *caspo-ts*, and we observe that the learned BNs only contain Boolean functions with clauses of size one. We also notice that the learned BNs are very similar to each other, as we see in Table 1 with the similarity score of 0.81. This relates to the fact that the ASP solver used by *caspo-ts* uses a backtracking algorithm to enumerate solutions and, hence, the solutions only change in small parts.

Next, we analyze the UACC812 cell line using *caspo-ts*^{D}. Fig. 5 shows the union of 10 TP BNs obtained by *caspo-ts*^{D}. Nodes, edges and colors have the same meaning as in Fig. 4. Unlike with *caspo-ts*, here we identified clauses with more than one element. They are represented by black rectangles where the nodes of incoming edges are their elements. Additionally, we use dashed edges to represent clauses that were also discovered by *caspo-ts*. In total, *caspo-ts*^{D} discovered 66 clauses, 41 more than *caspo-ts*. It identified 23 out of the 25 clauses discovered by the original system, and 43 additional clauses, studying the same number of BNs. It is important to note that even though for the UACC812 cell line *caspo-ts*^{D} learned 5 TP BNs less than *caspo-ts*, the number of clauses learned by *caspo-ts*^{D} is 3 times higher. Since we find much more clauses with *caspo-ts*^{D}, we can get an impression of the whole solution space by just inspecting a limited number of solutions. This analysis shows the efficacy of the extended *caspo-ts*^{D} system in a real case scenario, where it is difficult to study the complete solution space because of time constraints.

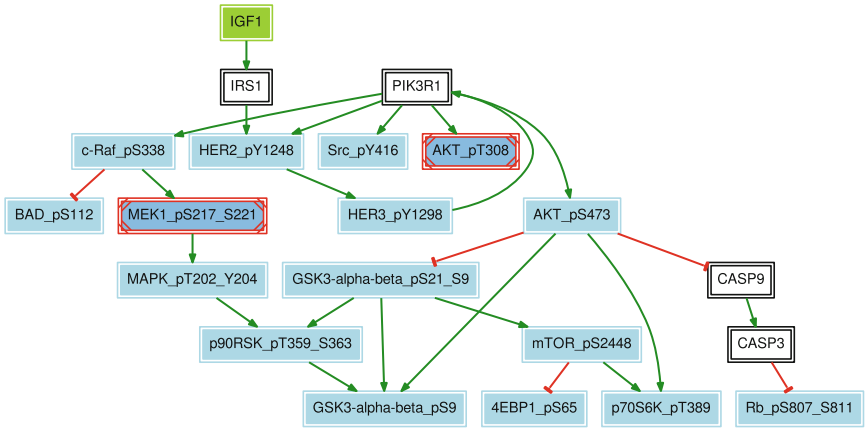

Fig. 4. *caspo-ts*: 10 optimal TPs BNs concatenated for cell line UACC812. All BNs are identically optimal. (Color figure online)

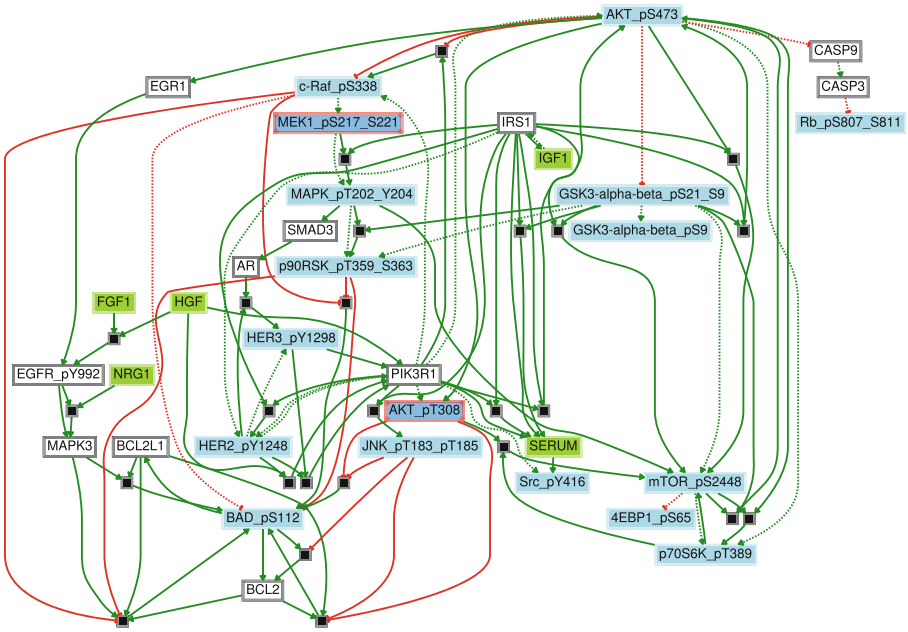

Fig. 5. *caspo-ts*^{D}: 10 optimal TP BNs concatenated for cell line UACC812. The dashed edges are used to represent clauses that were also discovered by *caspo-ts*. AND gates are represented by black boxes.

4 Conclusion

We have presented an algorithm to enumerate diverse optimal solutions with the *caspo-ts* system. The new algorithm extends the approach of [21] for computing diverse optimal solutions. The novelty of this extension is that by modifying the heuristic of the solver we manage to enumerate solutions that are both *optimal* and *diverse*. There are other approaches for computing diverse solutions [6,10,18] but they do not consider optimization problems. We applied *caspo-ts*^{D} on an artificial dataset (TCR signaling) as well as a real case study (HPN-DREAM) to learn diverse BNs. We compared the results with the *caspo-ts* system, showing a substantial improvement in solution quality. For one, we discovered more signaling behaviors (clauses) comparing solutions enumerated with both systems. For another, we were able to find solutions for cell line MCF7, where *caspo-ts* could not find solutions before. In the near future, we plan to extend the diversity algorithm in two directions. First, we are planning to experiment with solver parameters in order to introduce some randomness into the search. Second, we intend to extend the algorithm to call the model-checker only on answer sets which are diverse (according to some measure to be defined). This is possible because the time to enumerate over-approximated solutions using the ASP solver is much lower than the time needed to check solutions using the model-checker. We expect both enhancements to further improve the diversity of the discovered solutions.

References

1. Albert, R., Othmer, H.G.: The topology of the regulatory interactions predicts the expression pattern of the segment polarity genes in Drosophila melanogaster. J. Theor. Biol. **223**(1), 1–18 (2003)
2. Biane, C., Delaplace, F.: Abduction based drug target discovery using Boolean control network. In: Feret, J., Koeppl, H. (eds.) CMSB 2017. LNCS, vol. 10545, pp. 57–73. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-67471-1_4
3. Brewka, G., Delgrande, J., Romero, J., Schaub, T.: Implementing preferences with *asprin*. In: Calimeri, F., Ianni, G., Truszczynski, M. (eds.) LPNMR 2015. LNCS (LNAI), vol. 9345, pp. 158–172. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-23264-5_15
4. Calzone, L., et al.: Mathematical modelling of cell-fate decision in response to death receptor engagement. PLoS Comput. Biol. **6**(3), e1000702 (2010)
5. Carlin, D.E., et al.: Prophetic granger causality to infer gene regulatory networks. PloS one **12**(12), e0170340 (2017)
6. Eiter, T., Erdem, E., Erdoğan, H., Fink, M.: Finding similar or diverse solutions in answer set programming. In: Hill, P.M., Warren, D.S. (eds.) ICLP 2009. LNCS, vol. 5649, pp. 342–356. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02846-5_29
7. Gebser, M., Kaminski, R., Kaufmann, B., Schaub, T.: Clingo = asp + control: preliminary report. arXiv preprint arXiv:1405.3694 (2014)

8. Gelfond, M., Lifschitz, V.: The stable model semantics for logic programming. In: Kowalski, R., Bowen, K., (eds.) Proceedings of the Fifth International Conference and Symposium of Logic Programming (ICLP 1988), pp. 1070–1080. MIT Press (1988)
9. Guziolowski, C., et al.: Exhaustively characterizing feasible logic models of a signaling network using answer set programming. Bioinformatics **29**(18), 2320–2326 (2013)
10. Hebrard, E., Hnich, B., O'Sullivan, B., Walsh, T: Finding diverse and similar solutions in constraint programming. In: Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI 2005), pp. 372–377. AAAI Press (2005)
11. Hill, S.M., et al.: Inferring causal molecular networks: empirical assessment through a community-based effort. Nat. Methods **13**(4), 310–318 (2016)
12. Hill, S.M., et al.: Context specificity in causal signaling networks revealed by phosphoprotein profiling. Cell Syst. **4**(1), 73–83 (2017)
13. Kaminski, R., Schaub, T., Wanko, P.: A tutorial on hybrid answer set solving with *clingo*. In: Ianni, G., et al. (eds.) Reasoning Web 2017. LNCS, vol. 10370, pp. 167–203. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-61033-7_6
14. Kauffman, S.A.: The Origins of Order: Self-Organization and Selection in Evolution. Oxford University Press, Oxford (1993)
15. Klamt, S.A., Saez-Rodriguez, J., Lindquist, J.A., Simeoni, L., Gilles, E.D.: A methodology for the structural and functional analysis of signaling and regulatory networks. BMC Bioinf. **7**(1), 56 (2006)
16. MacNamara, A., Terfve, C., Henriques, D., Bernabé, B.P., Saez-Rodriguez, J.: State-time spectrum of signal transduction logic models. Phys. Biol. **9**(4), 045003 (2012)
17. Mitsos, A., Melas, I.N., Siminelakis, P., Chairakaki, A.D., Saez-Rodriguez, J., Alexopoulos, L.G.: Identifying drug effects via pathway alterations using an integer linear programming optimization formulation on phosphoproteomic data. PLoS Comput. Biol. **5**(12), e1000591 (2009)
18. Nadel, A.: Generating diverse solutions in SAT. In: Sakallah, K.A., Simon, L. (eds.) SAT 2011. LNCS, vol. 6695, pp. 287–301. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-21581-0_23
19. Ostrowski, M., Paulevé, L., Schaub, T., Siegel, A., Guziolowski, C.: Boolean network identification from perturbation time series data combining dynamics abstraction and logic programming. Biosystems **149**, 139–153 (2016)
20. Rau, A., Jaffrézic, F., Foulley, J.-L., Doerge, R.W.: An empirical Bayesian method for estimating biological networks from temporal microarray data. Stat. Appl. Genet. Mol. Biol. **9**(1) (2010)
21. Romero, J., Schaub, T., Wanko, P.: Computing diverse optimal stable models. In: OASIcs-OpenAccess Series in Informatics, vol. 52. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik (2016)
22. Rosenblueth, D.A., Muñoz, S., Carrillo, M., Azpeitia, E.: Inference of Boolean networks from gene interaction graphs using a SAT solver. In: Dediu, A.-H., Martín-Vide, C., Truthe, B. (eds.) AlCoB 2014. LNCS, vol. 8542, pp. 235–246. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07953-0_19
23. Sharan, R., Karp, M.: Reconstructing boolean models of signaling. J. Comput. Biol. **20**(3), 249–257 (2013)
24. Shmulevich, I., Dougherty, E.R., Zhang, W.: Gene perturbation and intervention in probabilistic Boolean networks. Bioinformatics **18**(10), 1319–1331 (2002)

25. Simons, P.: Extending the stable model semantics with more expressive rules. In: Gelfond, M., Leone, N., Pfeifer, G. (eds.) LPNMR 1999. LNCS (LNAI), vol. 1730, pp. 305–316. Springer, Heidelberg (1999). https://doi.org/10.1007/3-540-46767-X_22
26. Thakar, J., Albert, R.: Boolean models of within-host immune interactions. Curr. Opin. Microbiol. **13**(3), 377–381 (2010)
27. Videla, S., et al.: Revisiting the training of logic models of protein signaling networks with a ASP. In: Gilbert, D., Heiner, M. (eds.) CMSB 2012. LNCS, pp. 342–361. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-33636-2_20
28. Watterson, S., Marshall, S., Ghazal, P.: Logic models of pathway biology. Drug Discov. Today **13**(9), 447–456 (2008)
29. Wu, G., Dawson, E., Duong, A., Haw, R., Stein, L.: ReactomeFiviz: a Cytoscape app for pathway and network-based data analysis. F1000Research **3**, 146 (2014)