Scalable Enumeration of Trap Spaces in Boolean Networks via Answer Set Programming (supplementary material)

Van-Giang Trinh[1], **Belaid Benhamou**[1], **Samuel Pastva**[2], **Sylvain Soliman** [3]

[1]LIRICA team, LIS, Aix-Marseille University, Marseille, France
[2]Institute of Science and Technology, Klosterneuburg, Austria
[3]Lifeware team, Inria Saclay, Palaiseau, France
trinh.van-giang@lis-lab.fr, belaid.benhamou@lis-lab.fr, samuel.pastva@ist.ac.at, Sylvain.Soliman@inria.fr

Correctness

In this section, we formally prove Theorem 1 in the main text that represents the correctness of the proposed ASP encoding. We first define a Boolean Constraint Satisfiability Problem (CSP) as follows.

Definition 1. *Given a BN $\mathcal{N} = (V, F)$, the Boolean CSP $C(\mathcal{N})$ is the triple (R, D, C) where*

- $R = \{p_i \mid v_i \in V\} \cup \{n_i \mid v_i \in V\}$*, i.e., we have two variables for each node in V;*
- $D(p) = \mathbb{B}$ *for all $p \in R$, i.e., the variables are all Boolean;*
- C *includes the following constraints for each $v_i \in V$:*
  - $C_i^1 : p_i \vee n_i = 1$*;*
  - $C_i^2 : \widehat{f_i} = 1 \rightarrow p_i = 1$*, where* $\widehat{f_i} = NNF(f_i)\mid_{(p_1/v_1, n_1/\neg v_1), \ldots, (p_n/v_n, n_n/\neg v_n)}$*;*
  - $C_i^3 : \widehat{\neg f_i} = 1 \rightarrow n_i = 1$*, where* $\widehat{\neg f_i} = NNF(\neg f_i)\mid_{(p_1/v_1, n_1/\neg v_1), \ldots, (p_n/v_n, n_n/\neg v_n)}$.

We write $m(v_i) \geq f_i[m]$ iff for any trap space m, we have that if $f_i[m]$ can receive only 1 then $m(v_i) = 1$ or $m(v_i) = \star$, if $f_i[m]$ can receive only 0 then $m(v_i) = 0$ or $m(v_i) = \star$, and if $f_i[m]$ can receive both 0 and 1 (e.g., $f_i[m] = v_1$) then $m(v_i) = \star$. Clearly, by the characterization of trap spaces, m is a trap space of $\mathcal{N}$ iff $m(v_i) \geq f_i[m]$ for every $v_i \in V$.

Lemma 1. *Let $\mathcal{N} = (V, F)$ be a BN and $C(\mathcal{N})$ be the Boolean CSP of $\mathcal{N}$ following Definition 1. If both NNF(f_i) and NNF$(\neg f_i)$ are safe for all $v_i \in V$, then $C(\mathcal{N})$ is satisfied by a valuation r iff m is a trap space of $\mathcal{N}$ where for every $v_i \in V$ we have*

$$\begin{cases} m(v_i) = 1 & \text{iff } r(p_i) = 1 \wedge r(n_i) = 0, \\ m(v_i) = 0 & \text{iff } r(p_i) = 0 \wedge r(n_i) = 1, \\ m(v_i) = \star & \text{iff } r(p_i) = 1 \wedge r(n_i) = 1. \end{cases}$$

Proof. "$\Rightarrow$" If r is a satisfying valuation of $C(\mathcal{N})$, then m is a trap space of $\mathcal{N}$.

For every node $v_i \in V$, we have all three cases as follows. First, $r(p_i) = r(n_i) = 1$, implying $m(v_i) = \star$, leading to $m(v_i) \geq f_i[m]$ directly holds. Second, $r(p_i) = 1 \wedge r(n_i) = 0$, implying $m(v_i) = 1$. Since C_i^3 holds, we must have $\widehat{\neg f_i}(r) = 0$, since $r(n_i) = 0$. Recall that $\widehat{\neg f_i}$ has no negation following Definition 1. Clearly if NNF$(\neg f_i)[m]$ can receive 1 then $\widehat{\neg f_i}(r)$ always receives 1, thus NNF$(\neg f_i)[m] = 0$ must hold. Now, we have $\neg f_i[m] = 0$ because NNF$(\neg f_i)$ and $\neg f_i$ are logically equivalent, implying $f_i[m] = 1$, hence $m(v_i) \geq f_i[m]$ still holds. Third, $r(p_i) = 0 \wedge r(n_i) = 1$. Similar to the second case but considering $\widehat{f_i}$, we obtain that $m(v_i) \geq f_i[m]$ still holds. Overall, we can conclude that m is a trap space of $\mathcal{N}$. Note that in the forward direction, the safeness is not really relevant.

"$\Leftarrow$" If m is a trap space of $\mathcal{N}$, then r is a satisfying valuation of $C(\mathcal{N})$.

For every node $v_i \in V$, we have all three cases as follows. First, $m(v_i) = \star$, implying $r(p_i) = r(n_i) = 1$, leading to all the constraints of $C(\mathcal{N})$ directly hold. Second, $m(v_i) = 1$, implying $r(p_i) = 1 \wedge r(n_i) = 0$, leading to constraints C_i^1 and C_i^2 hold. Since m is a trap space, we have $m(v_i) \geq f_i[m]$, implying $f_i[m] = 1$ or equivalently $\neg f_i[m] = 0$. Then NNF$(\neg f_i)[m] = 0$ because NNF$(\neg f_i)$ and $\neg f_i$ are logically equivalent. Recall that $\widehat{\neg f_i}$ has no negation following Definition 1. For each input variable v_j of f_i, we have all three cases: $r(p_j) = 1 \wedge r(n_j) = 0$, $r(p_j) = 0 \wedge r(n_j) = 1$, and $r(p_j) = 1 \wedge r(n_j) = 1$. We can see that the first and second cases clearly do not affect the equivalence between $\widehat{\neg f_i}(r)$ and NNF$(\neg f_i)[m]$. In contrast, the third case raises a problem if NNF$(\neg f_i)$ is unsafe. However, since NNF$(\neg f_i)$ is safe, NNF$(\neg f_i)[m]$ cannot produce the formula in form of $v_j \wedge \neg v_j$ that is equal to 0, leading to valuation r really makes $\widehat{\neg f_i}$ equal to 0. Now, we have $\widehat{\neg f_i}(r) = 0$, hence C_i^3 holds. Third, $m(v_i) = 0$, implying $r(p_i) = 0 \wedge r(n_i) = 1$. Similar to the second case but considering the safeness of NNF(f_i), we obtain that all the constraints of $C(\mathcal{N})$ still hold. Overall, r is a satisfying valuation of $C(\mathcal{N})$. □

Lemma 2. *Any satisfying valuation of $C(\mathcal{N})$ (hence any trap space of $\mathcal{N}$) is represented by a Herbrand model of $\mathcal{L}$. In addition, any Herbrand model represents a satisfying valuation (hence a trap space), but multiple ones may represent the same satisfying valuation (hence the same trap space).*

Proof. It is easy to see that $\mathcal{L}$ has no negation and is logically

equivalent to the propositional formula characterizing the conjunction of all constraints of $C(\mathcal{N})$ via the introduction of some existentially quantified aux_k. Hence, for any satisfying valuation r of $C(\mathcal{N})$, we always have a corresponding Herbrand model I of $\mathcal{L}$ such that $p_i \in I$ iff $r(p_i) = 1$ and $n_i \in I$ iff $r(n_i) = 0$, for all $v_i \in V$. We also have a corresponding satisfying valuation r for any Herbrand model I such that $r(p_i) = 1$ iff $p_i \in I$ and $r(n_i) = 1$ iff $n_i \in I$. However, due to the introduction of some auxiliary atoms aux_k in $\mathcal{L}$, there may be other Herbrand models of $\mathcal{L}$ such that they all represent r. Such Herbrand models and also I have the same main atoms (p_i and n_i) and only differ in auxiliary atoms (aux_k). Since trap spaces of $\mathcal{N}$ one-to-one correspond to satisfying valuations of $C(\mathcal{N})$ by Lemma 1, we have the similar relation between trap spaces of $\mathcal{N}$ and Herbrand models of $\mathcal{L}$. □

By Lemma 2, any trap space of $\mathcal{N}$ is represented by a Herbrand model of $\mathcal{L}$. Hence, we define a recovering function $\gamma : \mathcal{S}^\star_\mathcal{N} \mapsto 2^\mathcal{A}$ such that $\gamma(m)$ includes the set of main atoms corresponding to trap space m and only the auxiliary atoms that are derived from these main atoms following the ASP rules of $\mathcal{L}$. Recall that $\mathcal{A}$ is the set of atoms of the encoded logic program $\mathcal{L}$ of $\mathcal{N}$. We can easily see that $\gamma(m)$ is the smallest set of atoms corresponding to trap space m. Also by Lemma 2, any Herbrand model of $\mathcal{L}$ represents a trap space of $\mathcal{N}$. Hence, we define a projection function $\pi : 2^\mathcal{A} \mapsto \mathcal{S}^\star_\mathcal{N}$ such that $\pi(I)$ is the trap space represented by the set of main atoms in Herbrand model I. Note that multiple Is may have the same projection.

Lemma 3. *For two any trap spaces m_1 and m_2 of $\mathcal{N}$, if $m_1 < m_2$, then $\gamma(m_1) \subset \gamma(m_2)$.*

Proof. Let M_1 and T_1 be the set of main atoms corresponding to m_1 and the set of auxiliary atoms derived from M_1, respectively. Let M_2 and T_2 be the set of main atoms corresponding to m_2 and the set of auxiliary atoms derived from M_2, respectively. Since $m_1 < m_2$, $M_1 \subset M_2$ by the correspondence between a Herbrand model of $\mathcal{L}$ and a trap space of $\mathcal{N}$. Consequently, $T_1 \subseteq T_2$ because $\mathcal{L}$ is positive. Since $\gamma(m_1) = M_1 \cup T_1$ and $\gamma(m_2) = M_2 \cup T_2$, we can conclude $\gamma(m_1) \subset \gamma(m_2)$. □

Theorem 1. *Let $\mathcal{N} = (V, F)$ be a BN and $\mathcal{L}$ be its encoded logic program. If both NNF(f_i) and NNF($\neg f_i$) are safe for all $v_i \in V$, then the set of stable models of $\mathcal{L}$ one-to-one corresponds to the set of minimal trap spaces of $\mathcal{N}$.*

Proof. Recall that the logic program $\mathcal{L}$ is disjunctive and positive. Hence, the set of stable models of $\mathcal{L}$ is equivalent to the set of subset-minimal ($\subset$-minimal) Herbrand models of $\mathcal{L}$ (Przymusinski 1991).

"$\Rightarrow$" If I is a stable model of $\mathcal{L}$, then $\pi(I)$ is a minimal trap space of $\mathcal{N}$.

Assume that $\pi(I)$ is not a minimal trap space of $\mathcal{N}$. Then there exists another trap space m of $\mathcal{N}$ such that $m < \pi(I)$. By Lemma 3, $\gamma(m) \subset \gamma(\pi(I))$. Recall that $\gamma(\pi(I))$ is the smallest set of atoms corresponding to trap space $\pi(I)$. Hence $\gamma(\pi(I)) = I$, leading to $\gamma(m) \subset I$, which is a contradiction because I is a $\subset$-minimal Herbrand model. Now, we can derive that $\pi(m)$ is a minimal trap space of $\mathcal{N}$.

"$\Leftarrow$" If m is a minimal trap space of $\mathcal{N}$, then $\gamma(m)$ is a stable model of $\mathcal{L}$.

Assume that $\gamma(m)$ is not a stable model of $\mathcal{L}$. Then there exists another Herbrand model I of $\mathcal{L}$ such that $I \subset \gamma(m)$. We cannot have $\pi(I) = m$ because $\gamma(m)$ is the smallest set of atoms corresponding to trap space m. Hence, $\pi(I) < m$, which is a contradiction because m is a minimal trap space. Now, we can derive that $\gamma(m)$ is a stable model of $\mathcal{L}$. □

Experimental evaluation

This section provides an extended description of all the performed experiments, their results, and the overall experimental setup.

Hardware and experiment configuration

Since our evaluation is primarily concerned with real-world performance, we first describe the exact hardware which was used to measure the presented results. Naturally, runtime measured on other hardware will differ. However, our configuration is fairly representative of a typical "mid-level" desktop available at the time of writing.

All experiments are performed using an 8-core Ryzen 5800X CPU fixed at 4.7Ghz (i.e. any turbo or overclocking is disabled to increase run-to-run consistency). The system is equipped with 128GB of DDR4-3200 memory at CAS latency of 18 cycles. The experiments are performed using a Debian 12 virtual machine such that each experiment is limited to approx. 64GB of RAM.

All compared tools are available as Python libraries. As such, our artifact[1] provides a Python virtual environment with all tools installed and configured (compatible with Debian 12), as well as the instructions and scripts for setting up this environment from scratch.

Experiments run sequentially (i.e. they do not compete for resources) and in general all use a single core (this is not limited in any way, it is simply the nature of the employed tools). The runtime is measured internally by each experiment script (i.e. it does not count the overhead of starting the Python interpreter), but otherwise includes all steps of the computation (e.g. loading the model, any conversions, transformations, or encoding, etc.).

Run-to-run consistency We have measured the run-to-run variance of each tool on a smaller subset of benchmarks. We observed only minimal differences: `<0.1s` for very short benchmarks (i.e. faster than `1s`) and less than `5%` of the runtime on longer running instances (details of the measurements are given in the artifact). Since our main conclusions are based on runtimes much larger than `1s`, and we test runtime across many benchmark models, each measurement is taken only once to reduce the total runtime of the benchmark suite (even at this point, the full benchmark suite

[1] https://zenodo.org/doi/10.5281/zenodo.10405520

runs for more than a week). For systems with higher run-to-run variance (e.g. laptops), the artifact supports reporting each experiment as an average of multiple repetitions.

Compared tools

We test a variety of tools which support the enumeration of trap spaces in BNs. In particular:

- Package `pyboolnet` is a tool based on (Klarner, Streck, and Siebert 2017). It computes trap spaces using prime implicants. We use the version `3.0.11`.[2]
- Package `mpbn` is a tool based on (Paulevé et al. 2020). It computes trap spaces of *locally-monotonic* models by direct ASP encoding. We use the version `1.7`.[3]
- Package `trappist` is a tool based on (Trinh et al. 2022). It provides a method for enumerating trap spaces based on Petri net siphons with ASP. We use the version `0.8`.[4]
- Package `trapmvn` is a tool presented in (Trinh et al. 2023). It iterates on `trappist` by adding heuristics which improve the Petri net encoding process. We use the latest version available on Github.[5]
- Package `tsconj` is the tool presented in this paper.

All these tools support enumeration of *minimal* trap spaces. To also test computation of *maximal* trap spaces, we have carefully introduced this feature into the packages that do not support it explicitly (i.e. `trappist` and `mpbn`). The source code of all tested packages, including our modifications, is available as part of the artifact.

Tested models

We test the tools on a wide range of real-world and synthetic models. In particular, we use:

Biodivine Boolean Models (BBM) The BBM repository (Pastva et al. 2023) provides a wide variety of published *biologically relevant* Boolean models. Here, we use the August 2022 edition[6] which consists of 212 models, ranging up to 321 variables and 1100 regulations.

Manually selected models To supplement the BBM dataset, we also performed a separate survey of related literature and found 39 additional *biologically relevant* models. A summary of these models, ranging up to 3158 variables, is given in Tables 1 and 2.

Very Large Boolean Networks (VLBN) The VLBN dataset[7] provides 28 *random* BNs with scale-free topology and inhibitor-dominant update functions. The models range up to 100.000 variables. They are very large, but have rather simple locally-monotonic update functions.

Manually generated models To supplement the VLBN dataset, we also generated 400 *random* BNs based on the generator provided in (Benes et al. 2021), ranging from 1000 to 5000 variables. This generator uses a degree distribution based on the BBM dataset to sample the network topology. For update functions, it samples from a subset of nested-canalizing locally-monotonic functions. Compared to the VLBN dataset, we thus cover a smaller range of sizes, but a wider range of update functions.

Complete results for minimal trap spaces

First, we test the enumeration of *minimal trap spaces*. Here, we test the time required to compute the *first* minimal trap space using each tool. Note that most models admit more than one trap space. In extreme cases, there can be millions or even billions of minimal trap spaces (although such situations are often biologically uninteresting in practice).

We find that enumerating many trap spaces often skews the experiment towards measuring the efficiency of the enumeration procedure and not the actual problem-solving. Hence, to control for this implementation aspect, we focus on the time necessary to obtain the first result.

Summary table The summary of the measured results is presented in Table 3. Here, we list the number of models for which a minimal trap space was found within the listed time. The `memory` column lists the number of models where the computation exceeded the memory limit of 64GB. The `time` column lists the number of models where the computation exceeded the time limit (10min for manually generated random models, one hour for all other models). Finally, the `other` column lists cases where the tool failed for some other reason. Specifically, for `mpbn`, this gives the number of non-locally-monotonic models in the dataset. For `trappist`, the tool sometimes failed when exceeding the recursion limit of the Python interpreter.

Summary figures To better visualize the performance scaling of individual tools on the random models, we prepared Figures 1 and 2. These figures show in detail how many models can be computed by each tool until a certain timeout. We do not have such figures for the real-world models, as the number of experiments running for more than one second is quite small in these datasets. Finally, Figure 3 shows the relative speed-up of `tsconj` compared to the other tools on *all* experiments (assuming both tools finished the experiment).

Discussion: real-world models First, let us observe that for both the BBM and the *selected* datasets, the differences between `tsconj` and `trappist` (or `trapmvn`) are minimal. All completed the vast majority of models in `<1s` and

[2] https://github.com/hklarner/pyboolnet/releases/tag/3.0.11

[3] https://github.com/bnediction/mpbn/releases/tag/v1.7; Note that newer versions of `mpbn` are available at the time of submission. However, version `2.0` does not modify the trap-space detection method, hence we skip it in our testing. Furthermore, very recently, version `3.0` was also published. However, the changes in this version are not officially published or documented yet. Due to this fact, and due to the relatively recent appearance of this version, we were not able to include it in our experiments.

[4] https://github.com/soli/trap-spaces-as-siphons/releases/tag/v0.8.0

[5] https://github.com/giang-trinh/trap-mvn/commit/69cc3cb5c8d7eb5db2cdf6add492f133cbedf3f9

[6] https://github.com/sybila/biodivine-boolean-models/releases/tag/august-2022

[7] https://doi.org/10.5281/zenodo.3714875

Table 1: Selected real-world models with < 200 variables. The table lists total variable count n, the number of source (i.e. input) variables s, and the respective bibliographic reference.

No.	Filename	n	s	Source
1	metastatic_melanoma	10	1	(Kadelka et al. 2020)
2	simplified_p53_high_dna_damage	16	0	(Kadelka et al. 2020)
3	simplified_p53_low_dna_damage	16	0	(Kadelka et al. 2020)
4	FT-GRN	23	5	(Chávez-Hernández et al. 2022)
5	DNA_damage_adaptation	26	1	(Kadelka et al. 2020)
6	Rho-family_GTPases_signaling	33	1	(Kadelka et al. 2020)
7	signaling-dependent-pluripotency	36	7	(Yachie-Kinoshita et al. 2018)
8	HOG_example	43	5	(Romers and Krantz 2017)
9	Pancreatic_Cancer	43	11	(Kadelka et al. 2020)
10	Drosophila	52	4	(Vega 2014)
11	mtor	59	0	(Hemedan, Schneider, and Ostaszewski 2023)
12	prkn_mitophagy	59	38	(Hemedan, Schneider, and Ostaszewski 2023)
13	dopamine_transcription	64	0	(Hemedan, Schneider, and Ostaszewski 2023)
14	hedgehog_signaling_pathway	65	28	(Kadelka et al. 2020)
15	foxo3	66	0	(Hemedan, Schneider, and Ostaszewski 2023)
16	ppargc1a	67	0	(Hemedan, Schneider, and Ostaszewski 2023)
17	EMT	69	8	(Rozum et al. 2021)
18	tca_cycle	69	35	(Hemedan, Schneider, and Ostaszewski 2023)
19	Bcell	73	5	(Dutta et al. 2019)
20	Executable_..._mast_cell_..._BCC	73	19	(Aghamiri et al. 2020)
21	Insulin_example	82	7	(Romers and Krantz 2017)
22	pi3k_akt	85	40	(Hemedan, Schneider, and Ostaszewski 2023)
23	Corral_ThIL17diff_15jan2021	92	16	(Corral-Jara et al. 2021)
24	EMT_Mechanosensing	136	4	(Sullivan et al. 2022)
25	macrophage_polarization	136	17	(Kadelka et al. 2020)
26	sprouting_angiogenesis	141	27	(Kadelka et al. 2020)
27	angiofull	142	28	(Weinstein et al. 2017)
28	EMT_Mechanosensing_TGFbeta	150	6	(Sullivan et al. 2022)
29	Executable_..._MAPK_model_BCC	181	37	(Aghamiri et al. 2020)

Table 2: Selected real-world models with > 200 variables. The table lists total variable count n, the number of source (i.e. input) variables s, and the respective bibliographic reference.

No.	Name	n	s	Source
1	Snf1_pathway	202	56	(Lubitz et al. 2015)
2	T-cell_co-receptor_molecules_calcium_channel	206	39	(Ganguli et al. 2015a)
3	YeastPheromoneModel	246	17	(Romers and Krantz 2017)
4	Mycobacterium_tuberculosis	317	37	(Kadelka et al. 2020)
5	Leishmania	342	81	(Ganguli et al. 2015b)
6	Executable_file_for_cholocystokinin_model_BCC	383	74	(Aghamiri et al. 2020)
7	ra_map	447	125	(Singh et al. 2023)
8	CAF-model	463	62	(Aghakhani et al. 2023)
9	Executable_file_for_Alzheimer_model_BCC	762	237	(Aghamiri et al. 2020)
10	Cell_cycle_control_network	3158	61	(Romers and Krantz 2017)

do not add any new results beyond <1min. The only practical difference is that trapmvn resolves the recursion issue in trappist and hence can solve three extra models. Similarly, mpbn also produces almost all solutions in <1s (and all in <1min), but is hindered by the lacking support for non-locally-monotonic models. Finally, while pyboolnet did not perform too poorly overall, it is clearly the slowest of the tested tools. Several models required much more than one minute, and it has the highest number of timeouts (8 in BBM and 6 in *selected*). Importantly, only tsconj was able to solve all problems in the BBM dataset, but neither tool found solutions for three of the *selected* models. In these three hard models, there are quite many very complicated unsafe formulas that have many input variables and

contain many nested sub-formulas, which is however not common in real-world models. That leads to a very big BDD for such a formula (regardless of the variable ordering within the BDD). We think this might be the source of hardness.

Discussion: random models For random models, the differences between the tools are much more apparent. First, as opposed to the smaller real-world models, `trappist`, `trapmvn` and `tsconj` actually vastly differ in their performance. Specifically, both `trappist` and `trapmvn` perform much worse than `tsconj`.

Here, the reason is the encoding process which uses BDDs to list the implicants of individual BN transitions. In larger models, the size of the BDD and the length of the implicant list can be substantial. This prevents the tool from encoding the problem into a logic program, or produces a program that is too large for the solver. The heuristics employed by `trapmvn` to mitigate this problem are to some extent effective: it solves 20 VLBN models as opposed to just one solved by `trappist`. In fact, for this dataset, it even outperforms `mpbn`. However, ultimately, both `trapmvn` and `trappist` are significantly outperformed by both `tsconj` and `mpbn` on the larger random dataset which admits more complex update functions.

When comparing `tsconj` to `mpbn`, the VLBN dataset clearly demonstrates the superior scalability of `tsconj`. Here, `tsconj` finished all models (most in `<1min`), while `mpbn` struggles once the model size exceed 50.000 variables. To validate that this improvement is not due to the limited scope of the VLBN dataset, we focus on the 400 generated random models. Here, both `mpbn` and `tsconj` completed *all* models. However, `tsconj` finished all models in `<1min`, while `mpbn` needed more than one minute for almost one hundred models (see also Figure 1). In fact, the geometric mean speedup of `tsconj` compared to `mpbn` in this dataset is 9.5× (1.6× min, 53.4× max). As such, our experiments show that `tsconj` is on average an order of magnitude better than `mpbn` in this setting.

Discussion: safe and unsafe formulas For all randomly generated models, every formula is safe. We observed that there are 138/251 (55%) real-world models containing unsafe formulas. More specifically, there are 3618/20210 (18%) unsafe Boolean formulas in all the real-world models. Overall, unsafe formulas are less common than safe ones, which is consistent with the common assumption in systems biology that the interactions between two components are monotonic (Paulevé et al. 2020).

Discussion: distribution of number of minimal trap spaces In general, the number of minimal trap spaces is closely tied to the number of attractors in asynchronous BNs. We are not aware of a study that would specifically target trap spaces, but the scaling of attractor count with respect to the network size has been thoroughly studied (Rozum et al. 2021). In our specific dataset, we observed that most real-world networks have only a small amount of trap spaces. In particular, more than 90% of networks have less than 10 minimal trap spaces.

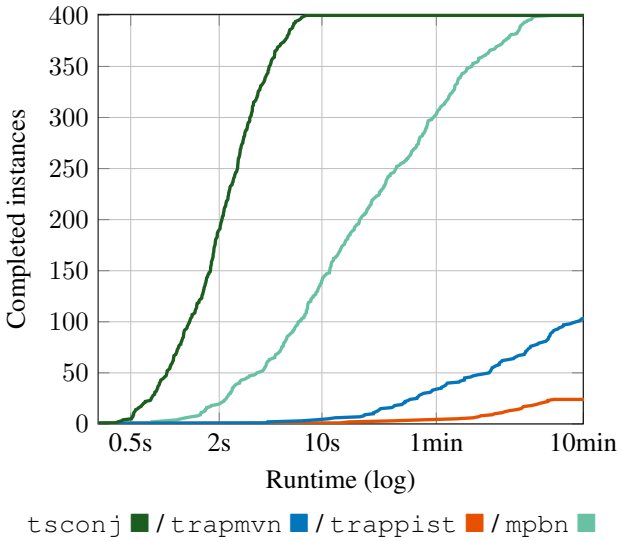

Figure 1: Cumulative **minimal** trap space experiments completed (y-axis) until a specific time point (x-axis, logarithmic). Concerns the 400 randomly generated models with 1.000-5.000 variables.

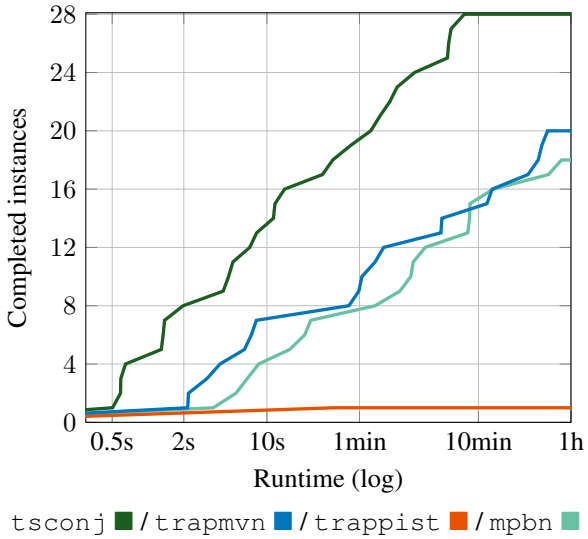

Figure 2: Cumulative **minimal** trap space experiments completed (y-axis) until a specific time point (x-axis, logarithmic). Concerns the 28 VLBN models with 1.000-100.000 variables.

Table 3: Summary of tool performance when computing the first **minimal** trap space. Rows `<1s`, `<1min` and `<1h` give the number of models completed within the respective time limit. Columns `memory`, `time` and `other` list the number of failed experiments in each category. Detailed explanation of each failure type is given in the main text.

Biodivine Boolean Models						
Method	<1s	<1min	<1h	memory	time	other
pyboolnet	161/212	194/212	203/212	1	8	0
mpbn	182/212	185/212	185/212	0	0	27
trappist	207/212	208/212	208/212	1	0	3
trapmvn	208/212	211/212	211/212	1	0	0
tsconj	208/212	212/212	**212**/212	0	0	0

Manually selected models						
Method	<1s	<1min	<1h	memory	time	other
pyboolnet	17/39	26/39	33/39	0	6	0
mpbn	26/39	27/39	27/39	0	1	11
trappist	35/39	36/39	36/39	2	1	0
trapmvn	35/39	36/39	36/39	2	1	0
tsconj	35/39	36/39	36/39	1	2	0

Very Large Boolean Networks						
Method	<1s	<1min	<1h	memory	time	other
pyboolnet	0/28	0/28	0/28	22	6	0
mpbn	0/28	7/28	18/28	0	10	0
trappist	0/28	1/28	1/28	0	27	0
trapmvn	0/28	9/28	20/28	5	3	0
tsconj	4/28	19/28	**28**/28	0	0	0

Manually generated models						
Method	<1s	<1min	<10min	memory	time	other
pyboolnet	0/400	0/400	0/400	0	400	0
mpbn	3/400	303/400	**400**/400	0	0	0
trappist	0/400	4/400	24/400	144	232	0
trapmvn	0/400	34/400	103/400	100	197	0
tsconj	67/400	400/400	**400**/400	0	0	0

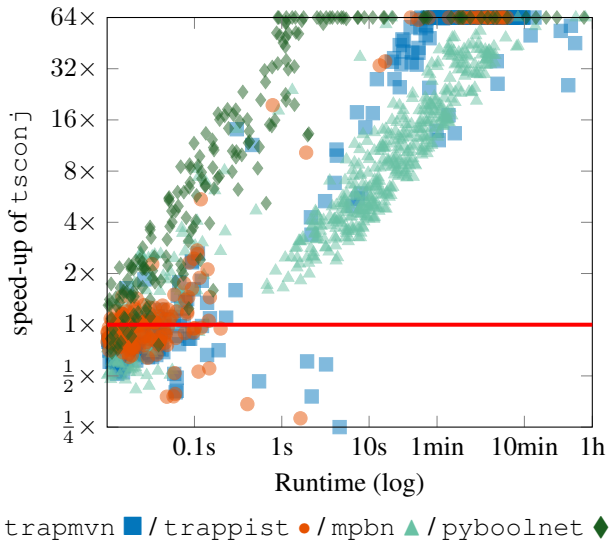

Figure 3: The speed-up of `tsconj` compared to the other tools on the **minimal** trap space problem, truncated to the interval $[\frac{1}{4}, 64]$. Points below the red line represent a slow-down instead of a speed-up.

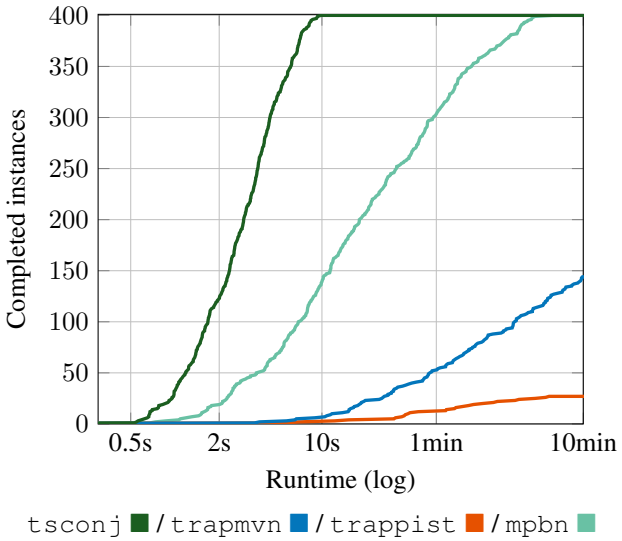

Figure 4: Cumulative **maximal** trap space experiments completed (y-axis) until a specific time point (x-axis, logarithmic). Concerns the 400 randomly generated models with 1.000-5.000 variables.

Complete results for maximal trap spaces

To provide a broader perspective on the performance of each tool, we also investigate the problem of *maximal* trap spaces. Note that (as mentioned), we had to modify `trappist` and `mpbn` to enable this feature. However, this modification is quite straightforward, since a simple adjustment of the logic program is sufficient.

The experimental settings are identical to those for the case of minimal trap space enumeration. The results are summarised in Figures 4, 5 6, as well as Table 4.

Overall, the results are largely in-line with the results for minimal trap spaces. However, in several instances the tools perform better because the problem is generally easier. Finally, note that amongst the selected models, there is one additional instance ($\star$) where `trappist` and `tsconj` report an error. This is because the model in question only has one trap space, corresponding to the whole state space. In such cases, the other tools return an empty result, but `tsconj` and `trappist` fail with an error.

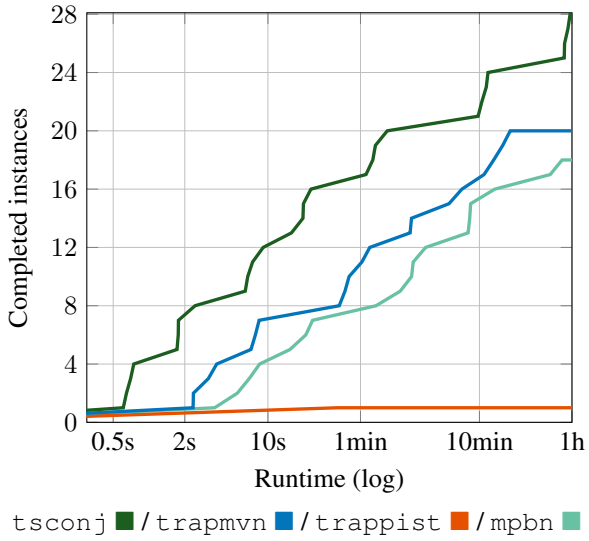

Figure 5: Cumulative **maximal** trap space experiments completed (y-axis) until a specific time point (x-axis, logarithmic). Concerns the 28 VLBN models with 1.000-100.000 variables.

References

Aghakhani, S.; Silva-Saffar, S. E.; Soliman, S.; and Niarakis, A. 2023. Hybrid computational modeling highlights reverse Warburg effect in breast cancer-associated fibroblasts. *bioRxiv*.

Aghamiri, S. S.; Singh, V.; Naldi, A.; Helikar, T.; Soliman, S.; Niarakis, A.; and Xu, J. 2020. Automated inference of Boolean models from molecular interaction maps using CaSQ. *Bioinform.*, 36(16): 4473–4482.

Benes, N.; Brim, L.; Pastva, S.; and Safránek, D. 2021. Computing Bottom SCCs Symbolically Using Transition Guided Reduction. In *International Conference on Computer Aided Verification*, 505–528. Springer.

Table 4: Summary of tool performance when computing the first **maximal** trap space. Rows `<1s`, `<1min` and `<1h` give the number of models completed within the respective time limit. Columns `memory`, `time` and `other` list the number of failed experiments in each category. Detailed explanation of each failure type is given in the main text.

Biodivine Boolean Models						
Method	<1s	<1min	<1h	memory	time	other
pyboolnet	161/212	194/212	203/212	0	9	0
mpbn	182/212	185/212	185/212	0	0	27
trappist	207/212	208/212	208/212	0	1	3
trapmvn	208/212	211/212	211/212	0	1	0
tsconj	208/212	212/212	212/212	0	0	0
Manually selected models						
Method	<1s	<1min	<1h	memory	time	other
pyboolnet	17/39	26/39	33/39	0	6	0
mpbn	26/39	27/39	27/39	0	1	11
trappist	34/39	35/39	35/39	0	3	1*
trapmvn	35/39	36/39	36/39	1	2	0
tsconj	34/39	35/39	35/39	1	2	1*
Very Large Boolean Networks						
Method	<1s	<1min	<1h	memory	time	other
pyboolnet	0/28	0/28	0/28	22	6	0
mpbn	0/28	7/28	18/28	0	10	0
trappist	0/28	1/28	1/28	0	27	0
trapmvn	0/28	10/28	20/28	4	4	0
tsconj	4/28	16/28	28/28	0	0	0
Manually generated models						
Method	<1s	<1min	<1h	memory	time	other
pyboolnet	0/400	0/400	0/400	0	400	0
mpbn	3/400	303/400	400/400	0	0	0
trappist	0/400	12/400	27/400	95	278	0
trapmvn	0/400	52/400	144/400	64	192	0
tsconj	32/400	400/400	400/400	0	0	0

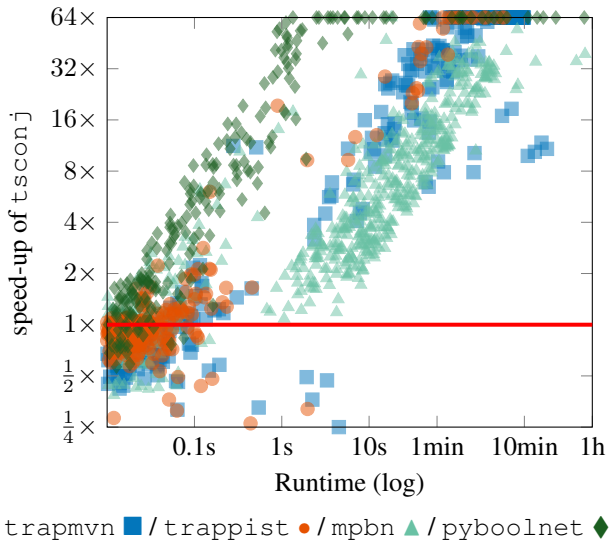

Figure 6: The speed-up of `tsconj` compared to the other tools on the **maximal** trap space problem, truncated to the interval $[\frac{1}{4}, 64]$. Points below the red line represent a slow-down instead of a speed-up.

Chávez-Hernández, E. C.; Quiroz, S.; García-Ponce, B.; and Álvarez-Buylla, E. R. 2022. The flowering transition pathways converge into a complex gene regulatory network that underlies the phase changes of the shoot apical meristem in Arabidopsis thaliana. *Front. Plant Sci.*, 13: 852047.

Corral-Jara, K. F.; Chauvin, C.; Abou-Jaoudé, W.; Grandclaudon, M.; Naldi, A.; Soumelis, V.; and Thieffry, D. 2021. Interplay between SMAD2 and STAT5A is a critical determinant of IL-17A/IL-17F differential expression. *Mol. Biomed.*, 2(1): 1–16.

Dutta, P.; Ma, L.; Ali, Y.; Sloot, P. M.; and Zheng, J. 2019. Boolean network modeling of B-cell apoptosis and insulin resistance in type 2 diabetes mellitus. *BMC Syst. Biol.*, 13(S2): 1–12.

Ganguli, P.; Chowdhury, S.; Bhowmick, R.; and Sarkar, R. R. 2015a. Temporal protein expression pattern in intracellular signalling cascade during T-cell activation: A computational study. *J. Biosci.*, 40(4): 769–789.

Ganguli, P.; Chowdhury, S.; Chowdhury, S.; and Sarkar, R. R. 2015b. Identification of Th1/Th2 regulatory switch to promote healing response during leishmaniasis: a computational approach. *EURASIP J. Bioinform. Syst. Biol.*, 2015: 13.

Hemedan, A. A.; Schneider, R.; and Ostaszewski, M. 2023. Applications of Boolean modeling to study the dynamics of a complex disease and therapeutics responses. *Front. Bioinf.*, 3: 1189723.

Kadelka, C.; Butrie, T.-M.; Hilton, E.; Kinseth, J.; and Serdarevic, H. 2020. A meta-analysis of Boolean network models reveals design principles of gene regulatory networks. *arXiv preprint arXiv:2009.01216*.

Klarner, H.; Streck, A.; and Siebert, H. 2017. PyBoolNet: a python package for the generation, analysis and visualization of Boolean networks. *Bioinform.*, 33(5): 770–772.

Lubitz, T.; Welkenhuysen, N.; Shashkova, S.; Bendrioua, L.; Hohmann, S.; Klipp, E.; and Krantz, M. 2015. Network reconstruction and validation of the Snf1/AMPK pathway in baker's yeast based on a comprehensive literature review. *npj Syst. Biol. Appl.*, 1(1): 1–10.

Pastva, S.; Safranek, D.; Benes, N.; Brim, L.; and Henzinger, T. 2023. Repository of logically consistent real-world Boolean network models. *bioRxiv*.

Paulevé, L.; Kolčák, J.; Chatain, T.; and Haar, S. 2020. Reconciling qualitative, abstract, and scalable modeling of biological networks. *Nat. Commun.*, 11(1): 1–7.

Przymusinski, T. C. 1991. Stable semantics for disjunctive programs. *New Gener. Comput.*, 9(3/4): 401–424.

Romers, J. C.; and Krantz, M. 2017. rxncon 2.0: a language for executable molecular systems biology. *BioRxiv*, 107136.

Rozum, J. C.; Gómez Tejeda Zañudo, J.; Gan, X.; Deritei, D.; and Albert, R. 2021. Parity and time reversal elucidate both decision-making in empirical models and attractor scaling in critical Boolean networks. *Sci. Adv.*, 7(29): eabf8124.

Singh, V.; Naldi, A.; Soliman, S.; and Niarakis, A. 2023. A large-scale Boolean model of the rheumatoid arthritis fibroblast-like synoviocytes predicts drug synergies in the arthritic joint. *npj Syst. Biol. Appl.*, 9(1).

Sullivan, E.; Harris, M.; Bhatnagar, A.; Guberman, E.; Zonfa, I.; and Regan, E. R. 2022. Boolean modeling of mechanosensitive Epithelial to Mesenchymal Transition and its reversal. *bioRxiv*.

Trinh, V.; Benhamou, B.; Hiraishi, K.; and Soliman, S. 2022. Minimal Trap Spaces of Logical Models are Maximal Siphons of Their Petri Net Encoding. In *International Conference on Computational Methods in Systems Biology*, 158–176. Springer.

Trinh, V.-G.; Benhamou, B.; Henzinger, T.; and Pastva, S. 2023. Trap spaces of multi-valued networks: definition, computation, and applications. *Bioinf.*, 39(Supplement_1): i513–i522.

Vega, M. R. 2014. Analyzing toys models of Arabidopsis and Drosphila using Z3 SMT-LIB. In *Independent Component Analyses, Compressive Sampling, Wavelets, Neural Net, Biosystems, and Nanoengineering XII*, volume 9118, 240–254. SPIE.

Weinstein, N.; Mendoza, L.; Gitler, I.; and Klapp, J. 2017. A Network Model to Explore the Effect of the Microenvironment on Endothelial Cell Behavior during Angiogenesis. *Front. Physiol.*, 8: 960.

Yachie-Kinoshita, A.; Onishi, K.; Ostblom, J.; Langley, M. A.; Posfai, E.; Rossant, J.; and Zandstra, P. W. 2018. Modeling signaling-dependent pluripotency with Boolean logic to predict cell fate transitions. *Mol. Syst. Biol.*, 14(1): e7952.