An Efficient Algorithm for Network Vulnerability Analysis under Malicious Attacks

T. Mancini, F. Mari, I. Melatti, I. Salvo, and E. Tronci

Computer Science Department, Sapienza University of Rome

Abstract. Given a communication network, we address the problem of computing a lower bound to the transmission rate between two network nodes notwithstanding the presence of an intelligent malicious attacker with limited destructive power.
Formally, we are given a link capacitated network N with source node s and destination node t and a budget B for the attacker.
We want to compute the Guaranteed Maximum Flow from s to t when an attacker can remove at most B edges. This problem is known to be NP-hard for general networks.
For Internet-like networks we present an efficient ILP-based algorithm coupled with instance transformation techniques that allow us to solve the above problem for networks with more than 200 000 nodes and edges within a few minutes. To the best of our knowledge this is the first time that instances of this size for the above problem have been solved for Internet-like networks.

1 Introduction

Given a communication network, especially an Internet sub-network, we are often interested in analysing its vulnerability to intelligent malicious attacks, consisting in removal (destruction) of network connections.

More specifically, given a communication network, we are interested in computing a lower bound to the transmission rate between two network nodes (*e.g.*, between a server and a router), notwithstanding the presence of a malicious attacker. Such attacker has limited *destructive power* B, but is intelligent enough to compute the maximum damage it can be provoked with B.

If, as usual, we model a communication network as a graph with edge capacities, the above problem becomes that of computing the Guaranteed Maximum Flow (GMF) when a malicious attacker can remove edges from the graph.

Of course, if an attacker can remove *any* number of edges, nothing can be guaranteed, that is the GMF is just zero. However, fortunately, even malicious attackers have a limited, possibly large, budget (*e.g.*, destructive power). This has motivated research on the computation of the GMF with a limited budget for the attacker. We will refer to this problem as *Network Interdiction Problem (NIP)* [34,39].

1.1 Motivations

Many networking problems can be cast as NIP (for a complete survey, we refer the reader to [36]). For example, as for network security, the following

problems have been cast as NIPs: DOS attacks [2], DOS-resistant authentication [3], and insider threat analysis [8]. Vulnerability analysis of infrastructural networks (such as electric power [9,24,26], water supply, critical infrastructure networks [32], etc.) can also be cast as a NIP [33,12,4]. Finally, NIP plays a role also in vulnerability-aware design of: communication networks [13], circuits [40], operating systems [37], and infrastructural networks [35].

1.2 Contributions

Unfortunately, NIP has been shown to be an NP-complete problem [34,39]. Nevertheless, many interesting NIPs can be solved casting NIP as an Integer Linear Programming (ILP) problem [39].

Here we focus on analysing vulnerability of meaningful Internet sub-networks, when some of their connections may be removed by an attacker. This entails solving NIPs for networks with hundreds of thousands of edges.

We propose an ILP-based algorithm to solve *very large* instances of the NIP which transforms the input instances in order to exploit the statistical structure of Internet-like networks.

Our experimental results (Section 4) show that, thanks to our NIP instance transformation techniques, our algorithm is able to analyse vulnerabilities of networks consisting of around 200 000 nodes in *a few minutes*. To the best of our knowledge, none of the previously proposed methods for NIP can handle Internet-like graphs of such size.

1.3 Paper outline

The paper is organised as follows. In Section 2 we give some preliminaries and formally state our Network Interdiction Problem. In Section 3 we outline our NIP instance transformation techniques which are the key enabler for the efficient vulnerability analysis of very large Internet-like networks using an ILP-based approach, along the lines of [39]. In Section 4 we show the effectiveness of our instance transformation techniques for the vulnerability analysis of realistic Internet-like networks of very large size.

2 Preliminaries

In the following we denote with $\mathbb{R}^+$ and $\mathbb{N}^+$ the set of positive real and natural numbers, respectively.

2.1 Capacitated networks

Here we recall the standard definitions of capacitated networks, flow, and maximum flow.

Definition 1 (Capacitated network). *A* directed [*undirected*] capacitated network *is a tuple* $N = (V, E, s, t, c)$ *where* (V, E) *is a directed* [*undirected*] *graph,* $s, t \in V$ $(s \neq t)$ *are, resp., the* source *and* destination *nodes and* $c : E \to \mathbb{R}^+$ *defines capacities of the links* E*. The capacity of a link* (u, v) *will be denoted by* $c(u, v)$ *or equivalently by* c_{uv}*.*

In the following we will use the shorter term *network* to refer to a capacitated network as defined in Definition 1.

Definition 2 (Network flow). *A* flow *in a network* $N = (V, E, s, t, c)$ *is a function* $f : E \to \mathbb{R}^+$ *s.t.* $\forall (i,j) \in E \; f(i,j) \leq c_{ij}$ *and,* $\forall j \in (V \setminus \{s,t\})$, $\sum_{i \in V|(i,j)\in E} f(i,j) = \sum_{i \in V|(j,i)\in E} f(j,i)$.

Definition 3 (Network flow value). *The* value *of a flow* f *in a network* $N = (V, E, s, t, c)$ *is* $\text{val}(f) = \sum_{v \in V|(s,v)\in E} f(s,v) = \sum_{v \in V|(v,t)\in E} f(v,t)$.

Definition 4 (Network maximum flow). *The* maximum flow *of a network* N *is* $\text{MaxFlow}(N) = \max\{\text{val}(f) | f \text{ is a flow in } N\}$.

2.2 Network Interdiction Problem

Here we formally define the problem we are interested in.

Definition 5 (Network Interdiction Problem, NIP). *A Network Interdiction Problem (NIP) is a triple* (N, B) *where* $N = (V, E, s, t, c)$ *is a network (directed or undirected) and* B *is a positive integer number (attacker* budget*).*

An (N, B) attack *is a map* α *from* E *to the set* $\{0, 1\}$ *s.t.* $\sum_{(u,v)\in E} \alpha(u,v) \leq B$. *We denote with* $N|_\alpha$ *the network* (V, E, s, t, c') *where* $c'(u,v) = (1-\alpha(u,v))c(u,v)$.

A solution *to the NIP* (N, B) *is an* (N, B) *attack* α *s.t., for all* (N, B) *attacks* β, $\text{MaxFlow}(N|_\alpha) \leq \text{MaxFlow}(N|_\beta)$.

If α *is a solution to the NIP* (N, B)*, we call* $\text{MaxFlow}(N|_\alpha)$ *the Guaranteed Maximum Flow (GMF) of* (N, B) *(notation:* $\text{GMF}(N, B)$*).*

In [39] it is proven that NIP is NP-hard. Various variants of the NIP have been proposed, envisioning, for example, the possibility for an attacker to reduce the capacity of edges (rather than destroying them completely), or different costs to destroy different edges (see, *e.g.*, [39,34]). In this paper, for space limitations and ease of presentation we focus on the core version of the problem. However, our approach can be generalised to most such variants.

2.3 An ILP formulation approach to NIP

Following the approach in [39], the NIP (N, B) for undirected networks $N = (V, E, s, t, c)$ can be formulated as the following Integer Linear Programming (ILP) problem.

$$
\begin{aligned}
\text{minimize} \quad & \sum_{(u,v)\in E} c_{uv} y_{uv} & & \\
\text{subject to:} \quad & \sum_{(u,v)\in E} z_{uv} \leq B & & \\
& y_{uv} + z_{uv} \geq d_u - d_v & \forall (u,v) \in E & \quad (1) \\
& y_{uv} + z_{uv} \geq d_v - d_u & \forall (u,v) \in E & \\
& d_s = 0, d_t = 1, d_u \in \{0,1\} & \forall u \in V \setminus \{s,t\} & \\
& y_{uv}, z_{uv} \in \{0,1\} & \forall (u,v) \in E &
\end{aligned}
$$

The main idea underneath such ILP problem is to compute a (s,t)-cut of N. Namely, the cut is identified by decision variables d_u (with value 1 if $u \in V$ is in the s partition and 0 otherwise) and y_{uv} (with value 1 if edge $(u,v) \in E$ crosses the cut and 0 otherwise).

Given this, the attack α, which is the solution to the instance of the NIP at hand, is defined as $\alpha(u,v) = z_{uv}$, for all $(u,v) \in E$, and the resulting GMF is $\text{MaxFlow}(N|_\alpha)$.

Note that, if the network N is directed, we just drop the set of constraints (1), as in directed networks flow goes only from s to t.

3 An Efficient NIP Algorithm on Internet-like networks

A direct use of state-of-the-art Integer Linear Programming (ILP) solvers (*e.g.*, GLPK, www.gnu.org/software/glpk, or CPLEX, www.ilog.com/products/cplex) to perform network vulnerability analysis allows us to tackle input network graphs consisting of at most a few tens of thousands of nodes. For very-large networks, the main focus of this paper, this direct approach is unviable.

In this section we present proper *instance transformation* techniques aimed at solving the Network Interdiction Problem (NIP) on very large network instances.

We point out that our goal is *not* to select classes of networks for which NIP is polynomial, but rather to *effectively* solve the NIP for the class of networks of our interest, *i.e.*, Internet-like networks. In particular, Internet-like networks typically do *not* fall in the classes of networks for which NIP is known to be polynomial, see, *e.g.*, [34]. Hence, our approach is much in the spirit with which Mixed Integer Linear Programming (MILP) and ILP solvers (see, *e.g.*, [14,28,15]), model checkers (see, *e.g.*, [5,17,20,21,27,23,22,38,18,25]), controllers synthesizers (see, *e.g.*, [1,6,19,29]), local search–based (see, *e.g.*, [10,31,16]), SAT and SMT solvers (see, *e.g.*, [30]) are built and exploited to solve large problem instances. Of course, being our problem NP-hard, our approach (if P $\neq$ NP) will still result in an exponential algorithm in the worst case. However, experimental results in Section 4 undoubtedly show that, by applying the instance transformation techniques outline next, *off-the-shelf* ILP solvers can be made able to perform vulnerability analysis of networks with 200 000 nodes in a just a few of minutes, with dramatic scalability improvements and huge speedups wrt. to a direct ILP encoding.

3.1 Structure of Internet-like networks

An *Internet-like* network is a network with a structure similar to the Internet graphs. In such networks, the distribution of node out-degrees (*i.e.*, the numbers of outgoing edges per node) is widely dispersed around the average value, in that most of the nodes have an out-degree much smaller than the average out-degree, and a few nodes (*hubs*) have an out-degree much larger than the average out-degree.

Our algorithm applies, in the given order, the following three network transformations to the input NIP instance. Such transformations are inspired to general networks processing techniques (see, *e.g.*, [7]), but have been carefully chosen and adapted to exploit the typical structure of Internet-like networks.

3.2 Phase 1: connected component selection

Given a NIP instance (N, B) over network $N = (V, E, s, t, c)$, our first step transforms it in (N', B), where the new network $N' = (V', E', s, t, c')$ is the single connected component of N containing both the source node s and the destination node t, and $c'(u, v) = c(u, v)$ for $(u, v) \in E'$.

Of course, in case s and t belong to two different connected components of N, the problem answer can be immediately computed, as the network Guaranteed Maximum Flow (GMF) is just zero.

3.3 Phase 2: detour elimination

Our second instance transformation step removes all *detour edges*, *i.e.*, edges which are *only* contained in cycles.

Formally, given our NIP instance (N', B) (as computed by the previous transformation step) with $N' = (V', E', s, t, c')$, we transform it in (N'', B), where network $N'' = (V'', E'', s, t, c'')$, with V'' induced by E'', $c''(u, v) = c'(u, v)$ for $(u, v) \in E''$ and:

$$E'' = \left\{ (u, v) \in E' \;\middle|\; \begin{array}{l} \text{exists a } (s,t)\text{-path in } N'' \text{ which} \\ \text{is not a cycle and contains edge } (u, v) \end{array} \right\}.$$

3.4 Phase 3: chains compaction

Our third and final transformation step focuses on maximal *chains*, *i.e.*, on the $\subseteq$-maximal non-singleton sets of nodes $\{v_1, \ldots, v_r\} \in V''$ $(r \geq 3)$ of network N'' such that, for each $1 \leq i < r$, $(v_i, v_{i+1}) \in E''$ and, for each $1 < i < r$ the out-degree of node v_i in N'' is 1. Our algorithm transforms network N'' by compacting each $\subseteq$-maximal chain $C = \{v_1, \ldots, v_r\} \subseteq V''$ in a single edge $(v_{\tilde{\jmath}(C)}, v_{\tilde{\jmath}(C)+1})$, being $\tilde{\jmath}(C) \in \{1, \ldots, r-1\}$ s.t. $(v_{\tilde{\jmath}(C)}, v_{\tilde{\jmath}(C)+1})$ is the edge with the minimum capacity of all the edges in the removed chain and $\tilde{\jmath}(C)$ is the minimum index with such property (in case the minimum is attained in more than one edge of the chain).

More formally, the compacted network $N''' = (V''', E''', s, t, c''')$ is such that V''' is induced by E''', $c'''(u, v) = c''(u, v)$ for $(u, v) \in E'''$ and:

$$\begin{aligned} E''' = (E'' \setminus \{(v_i, v_{i+1}) \in E'' \mid & \text{ exists a } \subseteq\text{-maximal chain } C = \{v_1, \ldots, v_r\} \subseteq V'', \\ & 1 \leq i < r, i \neq \tilde{\jmath}(C)\}) \\ \cup \{(v, v_{\tilde{\jmath}(C)}) \mid & \text{ exists a } \subseteq\text{-max. chain } C = \{v_1, \ldots, v_r\} \subseteq V'', (v, v_1) \in E''\} \\ \cup \{(v_{\tilde{\jmath}(C)}, v) \mid & \text{ exists a } \subseteq\text{-max. chain } C = \{v_1, \ldots, v_r\} \subseteq V'', (v_r, v) \in E''\} \end{aligned}$$

Our problem instance (N'', B) (output of the previous phase) is transformed into (N''', B).

Applying the above sequence of transformations to the input instance (N, B) at hand results in a new problem instance (N''', B). The following result shows that from a (optimal) solution for the latter we can efficiently compute (in linear time in the network size) a (optimal) solution to the former (proof omitted for lack of space).

Theorem 1. *Let (N, B) be an instance of a NIP, with $N = (V, E, s, t, c)$. Moreover, let (N''', B) be the NIP instance computed by applying the three transformations above to (N, B), with $N''' = (V''', E''', s, t, c''')$.*

Let α be a (optimal) solution to (N''', B). Then, $\beta(u, v) = \alpha(u, v)$ if $(u, v) \in E'''$ else 0 is a (optimal) solution to (N, B).

Summing up, our instance transformation techniques allow us to solve a NIP instance (N, B) by reducing it to (N''', B), by solving it through the ILP of Section 2.2, and by efficiently computing back an optimal solution to the original problem from the optimal solution computed for the latter.

4 Experiments

In order to assess effectiveness of the Network Interdiction Problem (NIP) instance transformation techniques of Section 3, below we report the experimental results obtained on very large sub-networks of the Internet. Our goal is to assess the *marginal impact* of our instance transformation techniques on the performance of an off-the-shelf Integer Linear Programming (ILP) solver.

All our experiments have been run on a 3GHz Intel Xeon QuadCore Linux PC with 8GB of RAM. Our instance transformation algorithms have been implemented in C. Our ILP solver is the well-known GLPK (www.gnu.org/software/glpk).

4.1 Defining NIP benchmark instances

Our first step is to generate realistic Internet-like networks of various sizes to be used as instances for our experiments. We started from an Internet snapshot G downloaded from [11]. Although G is a directed graph, for the aims of our experiments, we read edges of G as undirected (*i.e.*, bidirectional) edges.

Starting from G, we generated 20 sub-graphs G_e (where $e \in [1, 20]$) by randomly removing from G 20 given numbers of edges. Figure 1 shows statistical properties of our generated instances: for each graph, the figure shows its number of nodes and edges. Note that, for each $i \in [1, 19]$, graph G_{i+1} has both a higher number of nodes and a higher number of edges than graph G_i.

The step above only defines nodes and edges for our graphs. In order to translate them in networks, we have to define, for each graph, the edges capacity function c and the source and destination nodes s, t. As for c, we define $c(u, v)$ to be proportional to the degree of u ($\deg(u)$) and v ($\deg(v)$). More specifically, for each graph, we set $c(u, v) = 1000\frac{\deg(u)+\deg(v)}{\text{maxdeg}}$, where maxdeg is the maximum degree of a node in the graph. As for s and t, we randomly choose two distinct nodes among those having maximum degree in each graph.

As a result, we obtain 20 NIP instances. For each instance, value for B, defining the maximum number of links that the attacker can destroy (the attacker

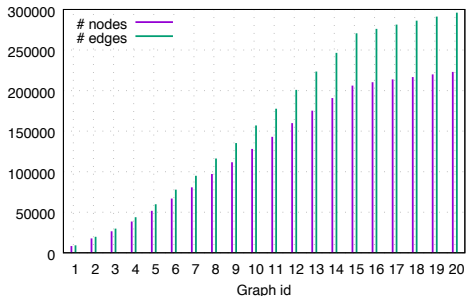


Fig. 1: Number of nodes and edges for our graphs $G_1, \ldots, G_{20}$.

budget), has been fixed fixed to the largest value for which the Guaranteed Maximum Flow (GMF) is greater than zero, thus placing us in the worst case (where the attacker has maximum freedom).

4.2 Experimental results

We are now ready to assess the *marginal impact* of our instance transformation techniques on the performance of the GLPK ILP solver.

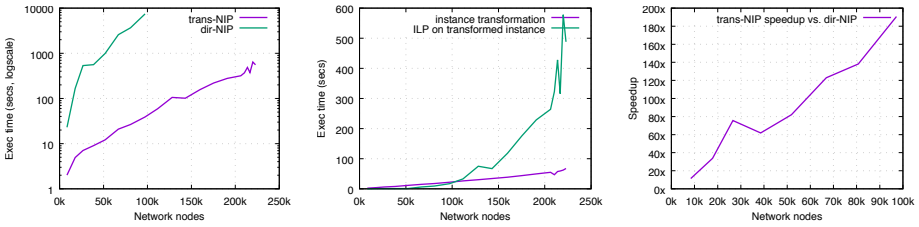
We solved each instance both by *directly* running the ILP problem defined in Section 2.2 (we call it the Direct NIP, dir-NIP, approach) and by first transforming the instance using our algorithm and then solving the ILP problem on the transformed instance (we call it the Transformed NIP, trans-NIP, approach).

For our comparisons we will focus on computation time, as memory requirements are always low (maximum 1GB of RAM). Furthermore, as in our networks the number of edges is nearly linear in the number of nodes (Figure 1), we will measure computation times as a function of the number of network nodes only.

Figure 2a shows the *dramatic* improvements that are *consistently* obtained by our approach (on *all* instances) based on applying the transformations of Section 3 before running our off-the-shelf ILP solver (trans-NIP). In the figure, the x axis shows the number of nodes of the networks $G_1, \ldots, G_{20}$ of Figure 1, and the y axis shows (in a logarithmic scale) the CPU computation time for trans-NIP and dir-NIP on each network.

Note that trans-NIP succeeds in completing the analysis and computing the GMF for *all* networks in our dataset in just *a few minutes*. On the contrary, dir-NIP only succeeds in computing (in times which are up to *2 orders of magnitude longer* than those of trans-NIP) the GMF for networks $G_1, \ldots, G_8$, which all have less than 100 000 nodes. On larger networks, GLPK did not even terminate after 24 hours (our timeout).

Note that, while CPU times for dir-NIP are just ILP solving times, CPU times for trans-NIP include both instance transformation and ILP solving times on the transformed networks. To assess the impact of our instance transformations in the overall trans-NIP computation time, Figure 2b shows a breakdown (for each instance) of the total trans-NIP solving time into instance transformation and



(a) (b) (c)

Fig. 2: (a) trans-NIP and dir-NIP computation times (seconds). (b) trans-NIP computation times breakdown. (c) trans-NIP speedup wrt. dir-NIP. All plots are drawn as functions of the number of nodes of our benchmark networks.

ILP solving time. The figure reveals that instance transformation indeed takes *negligible* time wrt. ILP solving.

Finally, Figure 2c shows the average speedup of trans-NIP wrt. dir-NIP on each network solved by both approaches (*i.e.*, $G_1, \ldots, G_8$). From the figure, we see that the speedups achieved by trans-NIP grow almost linearly with the number of nodes of the graph. Namely, the average speedup is at least 10× (for network G_1) and at most 190× (for network G_8), with an average of approximately 90×. This shows the feasibility and effectiveness of our approach on large and realistic Internet-like networks.

5 Conclusions

In this paper we focused on the Network Interdiction Problem (NIP) and presented effective instance transformation techniques to compute the Guaranteed Maximum Flow (GMF) of very large meaningful Internet-like networks, when an intelligent malicious attacker with limited resources tries to destroy network edges. Namely, our methodology adds suitable network transformations explicitly tailored to Internet-like networks to standard Integer Linear Programming (ILP)-based algorithms. Our experimental results show that our instance transformations are able to enable off-the-shelf ILP solvers (GLPK) to successfully analyse vulnerabilities of Internet sub-networks consisting of 200 000 nodes in just a few minutes, whilst a direct ILP application does not scale beyond around 100 000 nodes. In all our test cases, our methodology enables much faster analysis. In particular, on small networks, speedup wrt. direct ILP solving is at least 10×, growing up to 190× on the largest tested portions of Internet snapshots (consisting of around 200 000 nodes and 300 000 edges). On average, our algorithm is 90× faster than direct ILP solving. This shows the effectiveness of our approach.

Our instance transformation techniques are *explicitly designed* to exploit the *statistical properties* of Internet-like networks and might not by so effective on networks with a very different statistical structure. Finally, although we addressed the core version of the problem, our techniques can be generalised to most of the existing problem extensions and variants, as they mostly focus on the network structure.

Acknowledgements. This work was partially supported by the following research projects/grants: Italian Ministry of University & Research (MIUR) grant "Dipartimenti di Eccellenza 2018–2022" (Dept. Computer Science, Sapienza Univ. of Rome); EC FP7 project SmartHG (Energy Demand Aware Open Services for Smart Grid Intelligent Automation, 317761); INdAM "GNCS Project 2018".

References

1. V. Alimguzhin, F. Mari, I. Melatti, I. Salvo, and E. Tronci. Linearizing discrete time hybrid systems. *IEEE TAC*, 62(10), 2017.
2. T. Aura, M Bishop, and D. Sniegowski. Analyzing single-server network inhibition. In *Proc. CSFW 2000*, page 108. IEEE, 2000.
3. T. Aura, P. Nikander, and J. Leiwo. Dos-resistant authentication with client puzzles. In *Proc. Security Protocols Workshop 2000*. Springer, 2000.
4. R.L. Church, M.P. Scaparra, and R.S. Middleton. Identifying critical infrastructure: The median and covering facility interdiction problems. *Ann. Assoc. Am. Geogr.*, 94(3):491–502, 2004.
5. E. M. Clarke, O. Grumberg, and D.A. Peled. *Model Checking*. MIT Press, 1999.
6. G. Della Penna, B. Intrigila, D. Magazzeni, I. Melatti, and E. Tronci. Cgmurphi: Automatic synthesis of numerical controllers for nonlinear hybrid systems. *Eur. J. Control*, 19(1), 2013.
7. J. Evans. *Optimization Algorithms for Networks and Graphs*. Routledge, 2017.
8. D. Ha, S. Upadhyaya, H. Ngo, S. Pramanik, R. Chinchani, and S. Mathewa. *Insider Threat Analysis Using Information-Centric Modeling*, volume 242. Springer, 2007.
9. B.P. Hayes, I. Melatti, T. Mancini, M. Prodanovic, and E. Tronci. Residential demand management using individualised demand aware price policies. *IEEE Trans. Smart Grid*, 8(3), 2017.
10. H.H. Hoos and T. Stützle. *Stochastic Local Search: Foundations and Applications*. Elsevier, 2004.
11. The Internet Mapping Project: http://www.cheswick.com/ches/map/.
12. H. S. Jeong, J. Qiao, D.M. Abraham, M. Lawley, J.-P. Richard, and Y. Yih. Minimizing the consequences of intentional attack on water infrastructure. *Comp.-Aided Civil & Infrastructure Eng.*, 21:79–92, 2006.
13. T Korkmaz and M. Krunz. Multi-constrained optimal path selection. In *Proc. INFOCOM 2001*, pages 834–843, 2001.
14. Y. Lin, L.M. Austin, and J.R. Burns. An intelligent algorithm for mixed-integer programming models. *Comp. Oper. Res.*, 19(6):461–468, 1992.
15. T. Mancini. Now or Never: Negotiating efficiently with unknown or untrusted counterparts. *Fundam. Inform.*, 149(1-2), 2016.
16. T. Mancini, P. Flener, and J. Pearson. Combinatorial problem solving over relational databases: View synthesis through constraint-based local search. In *Proc. SAC 2012*. ACM, 2012.
17. T. Mancini, F. Mari, A. Massini, I. Melatti, F. Merli, and E. Tronci. System level formal verification via model checking driven simulation. In *Proc. CAV 2013*, volume 8044 of *LNCS*. Springer, 2013.
18. T. Mancini, F. Mari, A. Massini, I. Melatti, I. Salvo, S. Sinisi, E. Tronci, R. Ehrig, S. Röblitz, and B. Leeners. Computing personalised treatments through in silico clinical trials. A case study on downregulation in assisted reproduction. In *Proc. RCRA 2018*, 2018.
19. T. Mancini, F. Mari, A. Massini, I. Melatti, I. Salvo, and E. Tronci. On minimising the maximum expected verification time. *IPL*, 122, 2017.

20. T. Mancini, F. Mari, A. Massini, I. Melatti, and E. Tronci. Anytime system level verification via random exhaustive hardware in the loop simulation. In *Proc. DSD 2014*. IEEE, 2014.
21. T. Mancini, F. Mari, A. Massini, I. Melatti, and E. Tronci. System level formal verification via distributed multi-core hardware in the loop simulation. In *Proc. PDP 2014*. IEEE, 2014.
22. T. Mancini, F. Mari, A. Massini, I. Melatti, and E. Tronci. Anytime system level verification via parallel random exhaustive hardware in the loop simulation. *Microprocessors and Microsystems*, 41, 2016.
23. T. Mancini, F. Mari, A. Massini, I. Melatti, and E. Tronci. SyLVaaS: System level formal verification as a service. *Fundam. Inform.*, 1–2, 2016.
24. T. Mancini, F. Mari, I. Melatti, I. Salvo, E. Tronci, J. Gruber, B. Hayes, M. Prodanovic, and L. Elmegaard. Demand-aware price policy synthesis and verification services for smart grids. In *Proc. SmartGridComm 2014*. IEEE, 2014.
25. T. Mancini, F. Mari, I. Melatti, I. Salvo, E. Tronci, J.K. Gruber, B. Hayes, and L. Elmegaard. Parallel statistical model checking for safety verification in smart grids. In *Proc. SmartGridComm 2018*. IEEE, 2018.
26. T. Mancini, F. Mari, I. Melatti, I. Salvo, E. Tronci, J.K. Gruber, B. Hayes, M. Prodanovic, and L. Elmegaard. User flexibility aware price policy synthesis for smart grids. In *Proc. DSD 2015*. IEEE, 2015.
27. T. Mancini, E. Tronci, I. Salvo, F. Mari, A. Massini, and I. Melatti. Computing biological model parameters by parallel statistical model checking. In *Proc. IWBBIO 2015*, volume 9044 of *LNCS*. Springer, 2015.
28. T. Mancini, E. Tronci, A. Scialanca, F. Lanciotti, A. Finzi, R. Guarneri, and S. Di Pompeo. Optimal fault-tolerant placement of relay nodes in a mission critical wireless network. In *Proc. RCRA 2018*, 2018.
29. F. Mari, I. Melatti, I. Salvo, and E. Tronci. Model based synthesis of control software from system level formal specifications. *ACM TOSEM*, 23(1), 2014.
30. J. Marques-Silva and S. Malik. *Propositional SAT Solving*. Springer, 2018.
31. L. Michel and P. Van Hentenryck. *Constraint-Based Local Search*. Springer, 2017.
32. A.T. Murray and T.H. Grubesic. Critical infrastructure protection: The vulnerability conundrum. *Telemat. Inf.*, 29(1):56–65, 2012.
33. C. Phillips and L. Painton Swiler. A graph-based system for network-vulnerability analysis. In *Proc. NSPW 1998*, pages 71–79. ACM, 1998.
34. C.A. Phillips. The network inhibition problem. In *Proc. STOC 1993*, pages 776–785. ACM, 1993.
35. S. Shen. Optimizing designs and operations of a single network or multiple interdependent infrastructures under stochastic arc disruption. *Comput. Oper. Res.*, 40(11):2677–2688, 2013.
36. J.C. Smith, M. Prince, and J. Geunes. *Modern Network Interdiction Problems and Algorithms*, pages 1949–1987. Springer, 2013.
37. B. Tadayon and J.C. Smith. Algorithms and complexity analysis for robust single-machine scheduling problems. *J. Scheduling*, 18(6):575–592, 2015.
38. E. Tronci, T. Mancini, I. Salvo, S. Sinisi, F. Mari, I. Melatti, A. Massini, F. Davì, T. Dierkes, R. Ehrig, S. Röblitz, B. Leeners, T. H. C. Krüger, M. Egli, and F. Ille. Patient-specific models from inter-patient biological models and clinical records. In *Proc. FMCAD 2014*. IEEE, 2014.
39. R.K. Wood. Deterministic network interdiction. *Math. Comp. Mod.*, 17(2):1–18, 1993.
40. Y. Xiao, K. Thulasiraman, and G. Xue. Constrained shortest link-disjoint paths selection: A network programming based approach. *IEEE Trans. Circ. Sys.*, 53(5):1174–1187, 2006.