Université de Montréal

The Multilevel Critical Node problem : Theoretical Intractability and a Curriculum Learning Approach

par

Adel Nabli

Département d'informatique et de recherche opérationelle
Faculté des arts et des sciences

Mémoire présenté en vue de l'obtention du grade de
Maître ès sciences (M.Sc.)
en Informatique

Août 2020

© Adel Nabli, 2020

Université de Montréal

Faculté des arts et des sciences

Ce mémoire intitulé

The Multilevel Critical Node problem : Theoretical Intractability and a Curriculum Learning Approach

présenté par

Adel Nabli

a été évalué par un jury composé des personnes suivantes :

Emma Frejinger

(présidente-rapporteuse)

Margarida Da Silva Carvalho

(directrice de recherche)

Pierre l'Écuyer

(membre du jury)

Résumé

Évaluer la vulnérabilité des réseaux est un enjeu de plus en plus critique. Dans ce mémoire, nous nous penchons sur une approche étudiant la défense d'infrastructures stratégiques contre des attaques malveillantes au travers de problèmes d'optimisations multiniveaux. Plus particulièrement, nous analysons un jeu séquentiel en trois étapes appelé le « Multilevel Critical Node problem » (MCN). Ce jeu voit deux joueurs s'opposer sur un graphe : un attaquant et un défenseur. Le défenseur commence par empêcher préventivement que certains nœuds soient attaqués durant une phase de vaccination. Ensuite, l'attaquant infecte un sous ensemble des nœuds non vaccinés. Finalement, le défenseur réagit avec une stratégie de protection. Dans ce mémoire, nous fournissons les premiers résultats de complexité pour MCN ainsi que ceux de ses sous-jeux. De plus, en considérant les différents cas de graphes unitaires, pondérés ou orientés, nous clarifions la manière dont la complexité de ces problèmes varie. Nos résultats contribuent à élargir les familles de problèmes connus pour être complets pour les classes NP, Σ_2^p et Σ_3^p.

Motivés par l'insolubilité intrinsèque de MCN, nous concevons ensuite une heuristique efficace pour le jeu. Nous nous appuyons sur les approches récentes cherchant à apprendre des heuristiques pour des problèmes d'optimisation combinatoire en utilisant l'apprentissage par renforcement et les réseaux de neurones graphiques. Contrairement aux précédents travaux, nous nous intéressons aux situations dans lesquelles de multiples joueurs prennent des décisions de manière séquentielle. En les inscrivant au sein du formalisme d'apprentissage multiagent, nous concevons un algorithme apprenant à résoudre des problèmes d'optimisation combinatoire multiniveaux budgétés opposant deux joueurs dans un jeu à somme nulle sur un graphe. Notre méthode est basée sur un simple curriculum : si un agent sait estimer la valeur d'une instance du problème ayant un budget au plus B, alors résoudre une instance ayant un budget $B+1$ peut être fait en temps polynomial quelque soit la direction d'optimisation en regardant la valeur de tous les prochains états possibles. Ainsi, dans une approche ascendante, nous entraînons notre agent sur des jeux de données d'instances résolues heuristiquement avec des budgets de plus en plus grands. Nous rapportons des résultats quasi optimaux sur des graphes de tailles au plus 100 et un temps de résolution divisé par 185 en moyenne comparé au meilleur solutionneur exact pour le MCN.

Mots Clés : Optimisation Combinatoire, Optimisation Multiniveaux, Théorie de la Complexité, Hiérarchie Polynomiale, Apprentissage par Renforcement, Apprentissage Multiagent, Réseaux de Neurones Graphiques.

Abstract

Evaluating the vulnerability of networks is a problem which has gain momentum in recent decades. In this work, we focus on a Multilevel Programming approach to study the defense of critical infrastructures against malicious attacks. We analyze a three-stage sequential game played in a graph called the Multilevel Critical Node problem (MCN). This game sees two players competing with each other: a defender and an attacker. The defender starts by preventively interdicting nodes from being attacked during what is called a vaccination phase. Then, the attacker infects a subset of non-vaccinated nodes and, finally, the defender reacts with a protection strategy. We provide the first computational complexity results associated with MCN and its subgames. Moreover, by considering unitary, weighted, undirected and directed graphs, we clarify how the theoretical tractability or intractability of those problems vary. Our findings contribute with new NP-complete, Σ_2^p-complete and Σ_3^p-complete problems.

Motivated by the intrinsic intractability of the MCN, we then design efficient heuristics for the game by building upon the recent approaches seeking to learn heuristics for combinatorial optimization problems through graph neural networks and reinforcement learning. But contrary to previous work, we tackle situations with multiple players taking decisions sequentially. By framing them in a multi-agent reinforcement learning setting, we devise a value-based method to learn to solve multilevel budgeted combinatorial problems involving two players in a zero-sum game over a graph. Our framework is based on a simple curriculum: if an agent knows how to estimate the value of instances with budgets up to B, then solving instances with budget $B + 1$ can be done in polynomial time regardless of the direction of the optimization by checking the value of every possible afterstate. Thus, in a bottom-up approach, we generate datasets of heuristically solved instances with increasingly larger budgets to train our agent. We report results close to optimality on graphs up to 100 nodes and a 185× speedup on average compared to the quickest exact solver known for the MCN.

Keywords: Combinatorial Optimization, Multilevel Programming, Computational Complexity Theory, Polynomial Hierarchy, Reinforcement Learning, Multi-Agent Reinforcement Learning, Graph Neural Networks.

Contents

List of Tables

List of Figures

List of Acronyms & Abbreviations

CNDP	Critical Node Detection Problem
CNF	Conjunctive Normal Form
CNN	Convolutional Neural Network
CNP	Critical Node Problem
DAD	Defender-Attacker-Defender
DAG	Directed Acyclic Graph
DFS	Depth-First Search
DNF	Disjunctive Normal Form
DQN	Deep Q-Network
GNN	Graph Neural Network
MARL	Multi-Agent Reinforcement Learning

MBC	Multilevel Budgeted Combinatorial problems
MCN	Multilevel Critical Node problem
MDP	Markov Decision Process
MPs	Multilevel Programming problems
MVC	Minimum Vertex Cover
ReLU	Rectified Linear Unit
RL	Reinforcement Learning
RNN	Recurrent Neural Network

Acknowledgements

I would like to thank my parents for the way they cultivated my curiosity since my childhood, transmitted me the will to seek for academic excellence and supported me through my tortuous studies, I wouldn't have been able to come to Canada study at the Université de Montréal otherwise.

I address my deepest gratitude to Prof. Margarida Carvalho who supervised me wonderfully during my Master, I sincerely couldn't have dream of better and kinder supervisor. She introduced me to beautiful fields of Computer Science that I wouldn't have discovered otherwise, I am really grateful for that. She supported me when I had doubts about my future, always worked hard to provide me with great opportunities and advised me wisely. I really liked working with her, not only was it fun, but I also learnt a lot, and always felt I had the freedom to pursue my research interests.

Thus, I also would like to thank Céline Begin, who not only helped me a great deal with the administrative parts during the Master, but also oriented me towards Prof. Margarida when I was looking for a supervisor.

Finally, I send my deepest thanks and love to all my friends from Montréal. They sure illuminated my journey and provided me with the best environment to study, live and laugh there for two years. I would like to write my special thanks to all my roomates from "le 410": Lola Welsch, Nicolas Bégin, Emma Blanc, Zoé Delsalle and Margot Bréjard, as well as to the great friends from Centrale who shared the Montréal adventure with me: Alice Breton and Charlotte Dubost.

Introduction

Assessing the vulnerability of complex infrastructures such as networks is of utmost importance in practice. Indeed, many interconnected systems such as communication networks, electric grids or computer networks are vulnerable to malicious attacks or breakdowns. These malfunctions can be the origin of a cascade of failures spreading across the network, threatening vital parts of the system. Hence, detecting critical parts of networks and finding strategies to protect them against failures has attracted much attention in Computer Science, giving rise to several classes of problems such as the *Critical Node Detection problems* or *Interdiction Games on networks*. These questions being general, their applications are very diverse, ranging from floods control [Ratliff et al., 1975] to the decomposition of matrices into blocks [Furini et al., 2019a].

To minimize the propagation of a failure across a network, there are usually two kinds of strategies: *prevention*, which seeks to protect in advance the most critical parts of the system, and *blocking*, which aims at containing the harmful spread once it already began. In order to combine both, Baggio et al. 2020 introduced the Multilevel Critical Node problem (MCN), a zero-sum game between two players, a defender and an attacker. Given a graph $G = (V,A)$, the defender first vaccinates a set of nodes $D \subseteq V$ that become immune to infections, then the attacker attacks some other nodes $I \subseteq V \backslash D$, and, faced with the infection, the defender protects a new set of nodes $P \subseteq V \backslash (D \cup I)$, all moves restricted to budgets constraints. The overall contraction thus naturally leads to a trilevel optimization formulation, making the problem fall under the *Defender-Attacker-Defender* (DAD) framework introduced by Brown et al. 2006 to study the defense of critical infrastructure against malicious attacks.

Being both a trilevel optimization problem and related to provably hard network protection problems such as the *Firefighter problem* [Finbow et al., 2007] or the *Critical Node Problem* [Arulselvan et al., 2009], one can expect the MCN to be hard to solve. In practice, with a high-end CPU, the exact solver described in [Baggio et al., 2020] can take up to a few hours to solve instances with medium-sized graphs and only few units of budgets. Thus, there is a need to find new methodologies to speedup the process in order to tackle real-world instances of the problem, even if it means coming to terms with only finding sub-optimal solutions. One common way to do so is to design a scalable heuristic, at the expense of having no theoretical guaranty on the quality of the solutions produced.

In this thesis, we propose a framework to learn a heuristic to solve distributions of instances of the MCN problem, reporting results close to optimality on graphs up to 100 nodes and a $185\times$ speedup on average compared to the quickest exact solver know. First, we motivate our algorithmic approach by conducting an extensive study of the computational complexity of several variants of the MCN and of their subgames, managing to show that MCN is at least Σ_2^p-hard on directed graphs and Σ_3^p-complete on weighted ones. Then, leveraging recent advances in *Graph Neural Networks* and ideas from the emerging field of *Deep Learning for Combinatorial Optimization*, we frame the general class of *Multilevel Budgeted Combinatorial Optimization problems* to which the MCN belongs in a *Multi-Agent Reinforcement Learning* setting, allowing us to devise two meta-algorithms to learn to solve those problems, MultiL-DQN and MultiL-Cur. The former is a classic minimax procedure while the latter takes advantage of both the combinatorial setting *(the environment is deterministic)* and of the budgeted case *(the length of an episode is known in advance)* to create a curriculum, allowing our agent to gradually learn to solve increasingly harder sub-problems until it can tackle all the instances from the distribution of interest. We compare the two algorithms and experimentally demonstrate the efficiency of MultiL-Cur on 3 different versions of the MCN: the classic one tacking place on an undirected graph with unit weights on the nodes, the positively weighted case, and the directed case where the graph is directed.

We organize our discussion in the following way: first, in Chapter 1, we recall essentials in *Computational Complexity theory*, introducing definitions of some problem classes in the polynomial hierarchy. Then, in Chapter 2, we formally define the MCN problem, report its challenging nature in practice and present a detailed study of its complexity, as well as of its subgames. In Chapter 3, we define *Graph Neural Networks* in more details and introduce the field of *Reinforcement Learning*, allowing us to discuss previous work in *Deep Learning for Combinatorial Optimization*. Finally, in Chapter 4, we frame the general class of *Multilevel Budgeted Combinatorial problems* in the *Multi-Agent Reinforcement Learning* framework, devise several algorithms to learn heuristics to solve them and validate our methodology on different versions of the MCN.

Chapter 1

Essentials in Computational Complexity Theory

In many situations, algorithm designers are faced with problems so hard to solve that they seem intractable. However, proving the inherent intractability of a problem may be as difficult as finding an efficient algorithm to solve it. In this context, the theory of Computational Complexity emerged to arm practitioners with a way to characterize some of these seemingly hard problems, providing a mean to categorize problems by hardness class. In this chapter, we will introduce the classes of problems we will see throughout this thesis, beginning in Section 1.1 with the definition of NP-completeness and then presenting the Polynomial Hierarchy in Section 1.2, focusing on Σ_2^p-complete and Σ_3^p-complete problems.

1.1. NP-Completeness

In order to categorize problems by their hardness, we first need to formalize the setting. The problems we are considering are *decision problems*, *i.e.*, problems formulated in the form of a closed-ended question. Formally, a decision problem Π consists of a set of instances D_Π, which we can separate in two: $Y_\Pi \subseteq D_\Pi$ being the set of *Yes* instances, and $N_\Pi = D_\Pi \backslash Y_\Pi$ being the set of *No* instances. Given an instance of the decision problem, the aim is thus to find whether it is a *Yes* or *No* instance. In order to make this goal as clear as possible, we formulate our problems in two parts: the first describes what is a generic instance of the problem and the second asks the yes-no question in terms of the generic instance. For example, in the Dominating Set problem, we want to find a minimal sized subset $I \subseteq V$ of nodes of a graph $G = (V,E)$ such that every node not in I is adjacent to at least one member of I. The decision version of this problem is then stated as follows:

Dominating Set:
instance: A graph $G = (V,E)$ and a positive integer $K \leq |V|$.

QUESTION: Is there a subset $I \subseteq V$ with $|I| \leq K$ such that $\forall u \in V \backslash I$, $\exists v \in I$ $s.t.$ $(u,v) \in E$?

The problems Π for which there exists a deterministic algorithm able to answer the decision question for any instance in D_Π with a running time upper bounded by a polynomial expression in the size of the instance belong to the *polynomial complexity* class, denoted by P. The problems in this class are usually considered *easy* to solve from a computational complexity standpoint, even-though in practice having an algorithm running in $\mathcal{O}(n^{10})$, with n the size of the instance, becomes very quickly impractical.

Unfortunately, not all problems belong to P. Some, as the HALTING PROBLEM [Turing, 1936], have been proven to be *undecidable*, meaning that it is impossible to construct an algorithm that always leads to a correct yes-no answer. But between the class P and the undecidable one, there is a wide range of complexity classes. Among those, there is the *nondeterministic polynomial time* class, denoted by NP. Instead of asking to answer the decision question in polynomial time, NP contains problems Π where it is possible to *verify* in polynomial time that a solution proving that a given instance is in Y_Π is correct. For example, DOMINATING SET is in NP as, given G, K and a solution I, verifying that I makes it a *Yes* instance is doable in polynomial time. In particular, we directly have that P $\subseteq$ NP. However, whether this inclusion is strict or not is still an open question, even if it is believed that it is the case, meaning that there are problems in NP which cannot be solved *efficiently.*

Thus, if P $\neq$ NP, it is important to distinguish between problems in P and the ones in NP - P. To do so, we will introduce the notion of *NP-completeness* that derives from the definition of *polynomial transformation.* A polynomial transformation f is a function that maps a decision problem Π_1 to another decision problem Π_2 in polynomial time, such that any instance of the first problem is transformed into one of the second that has the same yes-no answer [Johnson, 2012]. Then, a problem Π is said to be *NP-hard* if, for any other problem $\Pi' \in$ NP, there exists a polynomial transformation f from Π' to Π. Moreover, Π is said to be *NP-complete* if $\Pi \in$ NP and Π is NP-hard. Thus, if P $\neq$ NP, all NP-complete problems are necessary in NP-P as if there existed one NP-complete Π in P, then all problems in NP would be solvable in polynomial time because there would exist a polynomial transformation between them and Π, contradicting the fact that P $\neq$ NP in the first place. Hence, we can think of NP-complete problems as *"the hardest problems in* NP" [Garey and Johnson, 1979].

Thus, one class of problems we will look into are NP-complete problems as they are provably intractable, assuming that P $\neq$ NP. To prove that a given decision problem Π is

NP-complete, we need to both show that it belongs to NP and that it is NP-hard. To show the latter, we need to exhibit a polynomial reduction to Π from any other problem $\Pi' \in$ NP. As the relation *"there exists a polynomial transformation from a problem to another"* is transitive, a sufficient way to do so is to find a polynomial transformation from another NP-hard problem. Thus, it is advantageous to look at problems that are already proven to be NP-hard. The first NP-complete result goes by the name of the *Cook-Levin Theorem* and showed the NP-completeness of the SATISFIABILITY problem [Cook, 1971; Levin, 1973]. In the following chapter, we will use one of its derivative, the 3-SATISFIABILITY (3-SAT) problem, proved to be NP-complete in the same paper by Cook [Cook, 1971] and whose decision version goes as follows:

3-SATISFIABILITY (3-SAT):
INSTANCE: Set U of variables, Boolean expression E over U in conjunctive normal form with exactly 3 literals in each clause $c \in C$.
QUESTION: Is there a 0-1 assignment for the variables in U that satisfies E ?

A few months later, Karp extended the number of known NP-complete problems to 21 [Karp, 1972], one of which being the KNAPSACK problem, which we will also use in our forthcoming reductions:

KNAPSACK:
INSTANCE: Finite set U, for each $u \in U$, a positive integer size a_u and a positive integer profit p_u, and two positive integers B and $\bar{K}$.
QUESTION: Is there a subset $U' \subseteq U$ such that $\sum_{u \in U'} a_u \leq B$ verifying $\sum_{u \in U'} p_u \geq \bar{K}$?

A few years later, Garey and Johnson published the first book about NP-completeness [Garey and Johnson, 1979], listing 300 NP-complete problems, one of which being the DOMINATING SET problem introduced earlier. Nowadays, there are thousands of problems proven to be NP-complete to chose from when trying to demonstrate that a new one is NP-hard.

1.2. The Polynomial Hierarchy

The polynomial hierarchy was introduced in [Meyer and Stockmeyer, 1972]. It allows to properly classify problems that appear to be harder than NP-complete. To understand what this means, and why it is a *hierarchy*, we take the example of the complexity class Σ_2^p which lies one level above the complexity class NP in the polynomial hierarchy. Σ_2^p is the class of decision problems that can be solved in polynomial time by a *nondeterministic* algorithm using an NP oracle, *i.e.*, an oracle outputing the correct answer to any NP decision problem in one computational step. In other words, if we use the notation A^B to denote the class of decision problems it is possible to answer in polynomial time using a machine of complexity A augmented with an oracle of complexity B, then $\Sigma_2^p = \mathrm{NP}^{\mathrm{NP}}$. Thus, in simple words, in the same way that NP-completeness gives insight on problems that may not be solvable in (deterministic) polynomial time, Σ_2^p-completeness categorizes those that remain that way even with access to an NP oracle. The first problem shown to be Σ_2^p-complete was the 2-ALTERNATING QUANTIFIED SATISFIABILITY (B_2) problem in the seminal work of Meyer and Stockmeyer 1972. If the boolean formula studied in B_2 is in *Disjunctive Normal Form* (DNF) with 3 literals per clause, then the problem is still Σ_2^p-complete [Wrathall, 1976]. Thus, if we consider expressions in *Conjunctive Normal Form* (CNF) with 3 literals per clause, instead of seeking to *satisfy* the boolean formula, we should state the question as formulated in [Johannes, 2011]:

2-CNF-ALTERNATING QUANTIFIED SATISFIABILITY (B_2^{CNF}):
INSTANCE: Two disjoint non-empty sets of variables X and Y and a Boolean expression E over $U = X \cup Y$ in conjunctive normal form with exactly 3 literals in each clause.
QUESTION: Is there a 0-1 assignment for X so that there is no 0-1 assignment for Y such that E is satisfied?

Through this example, we clearly see that the canonical Σ_2^p-complete problem is of the form $\exists X \forall Y \;\; \phi(X,Y)$ with $\phi(X,Y)$ a boolean predicate that can be evaluated in (deterministic) polynomial time. In fact, by definition, Σ_2^p contains all problems that can be stated that way.

But nothing holds us from adding more quantifiers. In fact, any decision problem of the form $\exists X_1 \forall X_2 \exists X_3 ... \forall X_k \;\; \phi(X_1, X_2, X_3, ... X_k)$ with k an integer is in Σ_k^p, and the decision problem asking whether or not it is possible to satisfy ϕ in such situation is named the k-ALTERNATING QUANTIFIED SATISFIABILITY problem (B_k), which has been claimed to be Σ_k^p-complete in [Meyer and Stockmeyer, 1972] and proved to be Σ_k^p-complete in [Wrathall, 1976]. Moreover, Wrathall 1976 showed that B_k is Σ_k^p-complete if ϕ is in CNF for odd k and in DNF for even k. In particular, the following decision problem $B_3 \cap \mathrm{CNF}$ is Σ_3^p-complete:

3-ALTERNATING QUANTIFIED SATISFIABILITY ($B_3 \cap \mathrm{CNF}$):
INSTANCE: Three disjoint non-empty sets of variables X, Y and Z, and a Boolean expression E over $U = X \cup Y \cup Z$ in conjunctive normal form with at most 3 literal in each clause $c \in C$.
QUESTION: Is there a 0-1 assignment for X so that for all 0-1 assignments of Y there is a

0-1 assignment of Z such that E is satisfied?

The polynomial hierarchy is thus defined recursively by setting Σ_0^p=P and $\Sigma_{k+1}^p = \mathrm{NP}^{\Sigma_k^p}$ for all integers $k \geq 0$. The name *polynomial hierarchy* PH stands for the union of every such Σ_k^p, *i.e.*, $\mathrm{PH} = \bigcup_{k \in \mathbb{N}} \Sigma_k^p$.

Contrary to the NP class for which thousands of problems have been proved to be complete nowadays, the list of known Σ_k^p-complete problems with $k \geq 2$ is still scarce. Indeed, in its 2008 version, Schaefer and Umans 2002 listed in their compendium no more than 80 different problems known to be complete for the second and higher levels of the polynomial hierarchy. Thus, extending the list of complete problems for those higher levels is of great interest.

Chapter 2

Complexity of the Multilevel Critical Node Problem

Parts of this chapter have appeared in the eponymous preprint "Complexity of the Multilevel Critical Node Problem" [Nabli et al., 2020] by Adel Nabli, Margarida Carvalho and Pierre Hosteins.

Contribution. *I was involved in all aspects of this scientific work: literature review, theoretical results (especially for the unitary and directed cases of the MCN), and writing.*

In the previous chapter, we introduced all the notions we use in this one to study the complexity of a three-stage sequential game played in a graph called the Multilevel Critical Node problem (MCN) [Baggio et al., 2020]. In this game, there are two players: a defender and an attacker. The defender starts by preventively interdicting nodes (vaccination) from being attacked. Then, the attacker infects a subset of non-vaccinated nodes and, finally, the defender reacts with a protection strategy. In this chapter, we first rigorously define the game by formulating it as trilevel mixed-integer program and name the variables we will refer to throughout this thesis. Second, we focus on how solvers for this problem behave in practice, which allows us to have a glimpse to the hardness of MCN and its subgames. Then, by considering unitary, weighted, undirected and directed graphs, we clarify how the theoretical tractability or intractability of those problems vary. Our findings contribute with new NP-complete, Σ_2^p-complete and Σ_3^p-complete problems.

In Section 2.1, we lay down the definition of the MCN. In Section 2.2 we report the computational challenges it raises for current exact solvers. Then, in Section 2.3, we detail our findings about the computational complexity of the MCN.

2.1. The Multilevel Critical Node Problem

Graphs are powerful mathematical structures that enable us to model real-world networks. The problem of breaking the connectivity of a graph has been extensively studied in combinatorial optimizationg since it can serve to measure the robustness of a network to disruptions. In this thesis, we focus on the Multilevel Critical Node problem (MCN) [Baggio et al., 2020]. Let $G = (V,A)$ be graph with a set V of nodes and a set A of arcs. In MCN there are two players, designated by defender and attacker, whose individual strategies are given by a selection of subsets of V. The game goes as follows: first, the defender selects a subset of nodes $D \subseteq V$ to *vaccinate* subject to a budget limit Ω and a cost $\{\hat{c}_v\}_{v\in V}$; second, the attacker observes the vaccination strategy, and selects a subset of nodes $I \subseteq V \setminus D$ to *infect* subject to a budget limit Φ and a cost $\{h_v\}_{v\in V}$; and third, the defender observes the infection strategy, and selects a subset of nodes $P \subseteq V \setminus I$ to *protect* subject to a budget limit Λ and a cost $\{c_v\}_{v\in V}$. Infected nodes propagate the infection to their neighbourhood, except to vaccinated or protected nodes. The goal of the defender is to maximize the benefit b_v, of saved nodes (*i.e.*, not infected), while the attacker aims to minimize it. We assume that all parameters of the problem are non-negative integers. The game description can be succinctly given by the following trilevel program:

$$(MCN) \quad \max_{\substack{z\in\{0,1\}^{|V|} \\ \sum_{v\in V} \hat{c}_v z_v \leq \Omega}} \quad \min_{\substack{y\in\{0,1\}^{|V|} \\ \sum_{v\in V} h_v y_v \leq \Phi}} \quad \max_{\substack{x\in\{0,1\}^{|V|} \\ \alpha\in[0,1]^{|V|}}} \quad \sum_{v\in V} b_v \alpha_v \tag{1}$$
$$\begin{aligned} \sum_{v\in V} c_v x_v &\leq \Lambda && \\ \alpha_v &\leq 1 + z_v - y_v && \forall v \in V \\ \alpha_v &\leq \alpha_u + x_v + z_v && \forall\ (u,v) \in A \end{aligned}$$

where $D = \{v \in V : z_v = 1\}$, $I = \{v \in V : y_v = 1\}$ and $P = \{v \in V : x_v = 1\}$. To gain more intuition about the setting, we provide in Figure 2.1 a simple example of an MCN game played on a directed graph where all the costs and benefits are unitary.

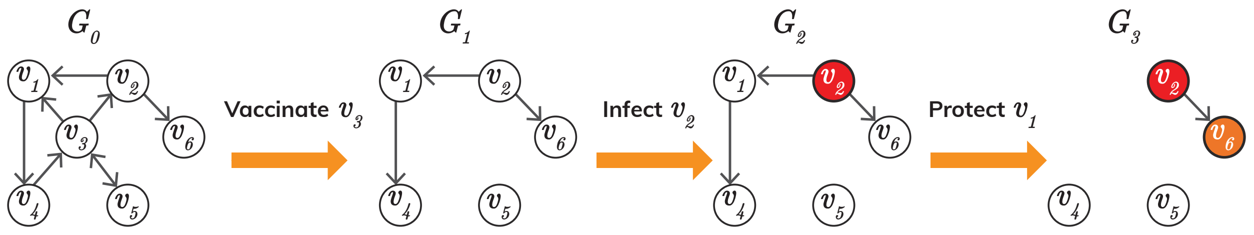


Figure 2.1. Example of an MCN game on a directed graph with unitary costs and benefits and budgets $\Omega = \Phi = \Lambda = 1$. We removed the vaccinated and protected nodes, see Property 2.3.1 for a justification. Here, $\{v_1,v_3,v_4,v_5\}$ are saved and $\{v_2,v_6\}$ are infected in the end.

2.2. In practice: behaviour of exact solvers for the MCN

More than introducing the MCN problem, Baggio et al. 2020 also provided an exact solver for the unitary case where costs and benefits are all set to one. In the MCN, the three levels share the same objective function, up to the direction of optimization. Thus, in the unitary case, it is possible to equivalently reformulate (1) as a single-level program and then use a Mixed Integer Linear Programming (MILP) solver to find an exact solution to any given instance of MCN. In order to do that, Baggio et al. 2020 first relaxed the integer requirement of the third level variable x and reformulated the last two levels of the problem, corresponding to the attack phase and the protection one, into a MILP using strong duality and the McCormick convex relaxation [McCormick, 1976] to linearize the bilinear terms. This relaxed linearized bilevel problem is named rlxAP. The algorithm using rlxAP as a subroutine to find an (integer) optimal solution to the attack and protection problem given a vaccination strategy is named AP. To solve MCN, Baggio et al. 2020 reformulated the trilevel optimization formulation into a MILP named 1lvlMIP, but relaxed the feasible region by limiting the attacks admissible. Then, they gradually restricted the feasible region by taking into account more and more attacks using AP as a subroutine, until finding the solution to MCN, see [Baggio et al., 2020] for details on the procedure.

2.2.1. Solving the MCN

In this part, we focus on the behaviour of a solver for MCN that uses the method described earlier. For that, we rely on the results of Baggio et al. 2020 as they already undertook this study. Using IBM ILOG CPLEX 12.7.1.0 on a single core of an Intel Xeon E5-2637 processor clocked at 3.50GHz with 8 GB RAM, they reported that the average time necessary for the best solver they considered (MCN^{MIX}) to solve instances of MCN on graphs with 100 nodes is 848 seconds. The particularity of MCN^{MIX} being that it uses the procedure MIX ++ of [Fischetti et al., 2017] to directly solve the bilevel integer problem AP to optimilaty without relying on the relaxed version rlxAP. The instances of MCN Baggio et al. 2020 considered being ones constituted of Erdős-Rényi undirected random graphs [Erdos and Renyi, 1960] $G = (V,E)$ with $|V| \in \{20,40,60,80,100\}$ and edge density between 5% and 15% with budgets Ω, Φ, Λ in $[\![1,3]\!]$. In order to gain more insight into their results, we gathered all the statistics produced by Baggio et al. 2020 for their paper, the list of instances they used and times they took to solve them being publicly available[1]. In Figure 2.2 we give more details into the evolution of the average time taken to solve those instances with the number of nodes of the graphs considered. In Table 2.1, we detail the number of solved instance, given that they stopped trying to solve instances if it took more than 2 hours (*i.e.*, the time we display

1. `https://github.com/mxmmargarida/Critical-Node-Problem`

in Figure 2.2 is actually the average time among all instances that took less than 2 hours to solve).

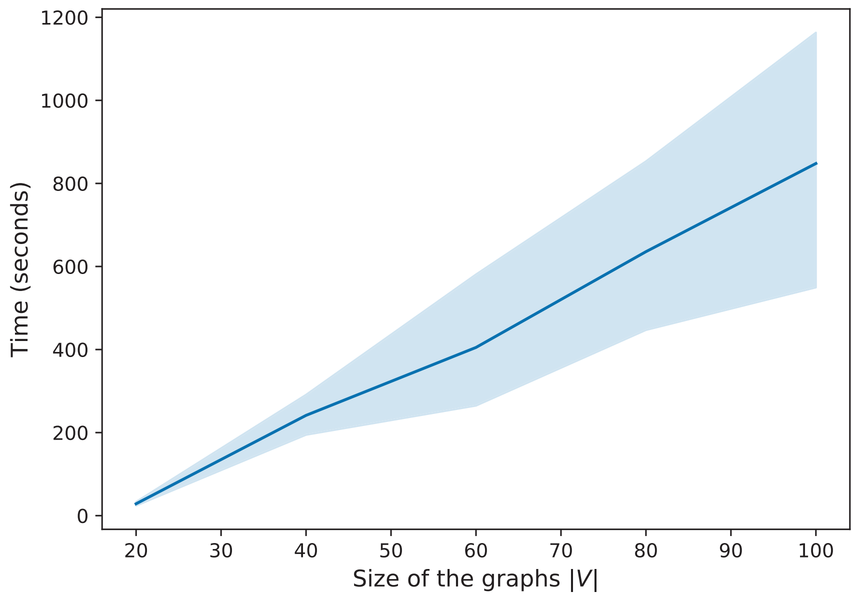


Figure 2.2. Evolution of the time taken by the exact solver MCN^{MIX} to solve instances of MCN with respect to the number of nodes of the graphs considered.

$\|V\| =$	20	40	60	80	100
Number of instances given to the solver	120	1080	120	120	120
Number of instances solved under 2 hours	120	876	110	101	85
Fraction of instances solved under 2 hours (%)	100	81	92	84	71

Table 2.1. Fraction of instances solved under 2 hours by MCN^{MIX}.

Thus, disregarding that the instances reaching the time limit are unaccounted for, what Figure 2.2 seems to tell us is that there is a linear relationship between the time necessary to solve an instance of MCN with MCN^{MIX} and its size. However, the numbers in Table 2.1 complicate the story. Indeed, the larger the size of the graphs, the more instances were disregarded by the solver for computation time reasons. Thus, in order to correct this bias, we decided to reproduce Figure 2.2, but this time with the addition of 2 hours entries in the lists of times corresponding to a lower bound on the time it would have been necessary to actually solve the disregarded instances. Moreover, in Table 2.1, we notice that the number of instances with 40 nodes is larger than any other. Indeed, Baggio et al. 2020 undertook a detailed study of the graphs of size 40, varying the edge density for this size only. Thus, to correct the bias, we disregarded all entries corresponding to graphs of size 40 with edge density different than 5%, the edge density used to generate all the instances of other size. The corrected plot is displayed in Figure 2.3.

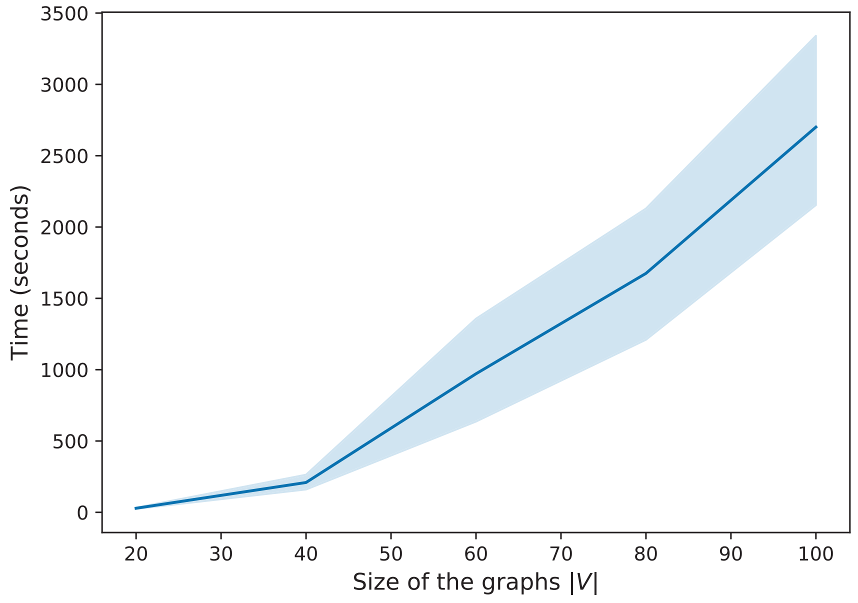
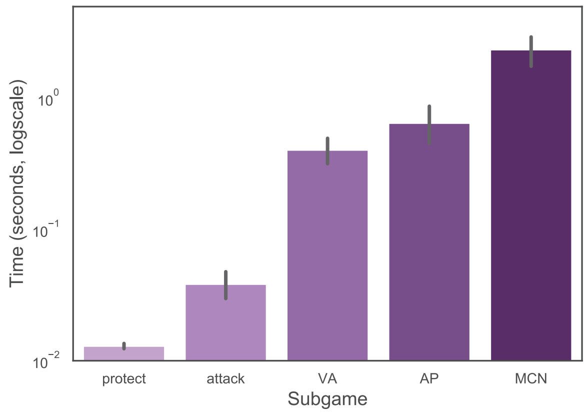


Figure 2.3. Corrected version of Figure 2.2

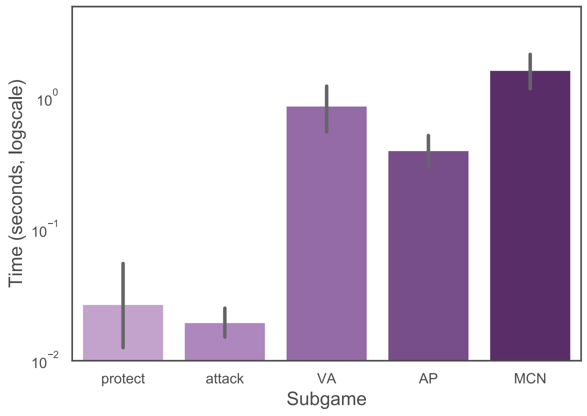
Remembering that Figure 2.3 is produced using only *lower bounds* on the times necessary to solve the hardest instances for MCN^{MIX}, we remark that, contrary to what we could conclude from Figure 2.2, the relationship between the size of the instances of MCN and the time necessary to solve them seems more *exponential*, which could suggest that MCN $\notin$ P or that MCN^{MIX} is not efficient to solve MCN.

2.2.2. Solving subgames of the MCN

So far, we focused our study on the behaviour of an exact solver for the whole trilevel problem. However, MCN is actually constituted of several subgames. In this part and later, we investigate the subgames *(i)* Protect, where given the vaccination strategy D and the attack one I, the defender seeks the optimal protection strategy, *(ii)* Attack, where given D and no protection budget, the attacker determines the optimal infection strategy, *(iii)* Attack-Protect, where given D, the attacker computes the optimal infection strategy, and *(iv)* Vaccination-Attack, where given no budget for protection, the defender finds the optimal vaccination strategy. Here, we compare the times necessary to solve each of the subgames of MCN. To do that, we re-implemented the method of Baggio et al. 2020 described earlier. Although it is originally designed for undirected graphs, all the formulations and algorithmic approaches also hold for directed ones. Thus, we run our algorithms in the unitary setting for both directed and undirected graphs. In Figure 2.4, we report results averaged over 50 Erdős-Rényi random graphs [Erdos and Renyi, 1960] of size 20 and with edge density of 15% for each subgame. All of the budgets Ω, Φ, Λ are random integers in $[\![1,3]\!]$. The results are generated using IBM ILOG CPLEX 12.9.0 on my laptop, which has an Intel Core i7-6700 HQ CPU clocked at 2.60 GHz and 16 GB of RAM.



(a) Undirected case



(b) Directed case

Figure 2.4. Time necessary to solve MCN and its subgames on graphs of size 20.

Thanks to Figure 2.4, we notice that there are roughly 3 categories of problems in both directed and undirected cases, with a hierarchy coinciding perfectly with the number of levels in each subgame. The quickest problems to solve are PROTECT and ATTACK, VACCINATION-ATTACK and ATTACK-PROTECT seem much more expensive, but the most time consuming problem of all is MCN in both cases. Thus, one can expect to find such a hierarchy in the computational complexities of the different games.

2.3. Complexity results for the MCN

In this section, we provide a computational complexity classification of the decision version of MCN, as well as, of its subgames. This fundamental contribution sheds light on the practical difficulties we noted in the last section. Furthermore, it contributes to the understanding of sequential combinatorial games within the polynomial hierarchy and motivates the focus on potentially $\Omega(2^{2^{2^{|V|}}})$ algorithms, heuristic methods or novel solution definitions. Table 2.2 summarizes our results; unitary cases assume that all costs and benefits are 1.

In Section 2.3.1, we revise the literature associated with MCN, allowing to position our contribution in the context of Critical Node problems. In Section 2.3.2, we focus on the case where graphs are undirected and each node benefit and cost is unitary. Section 2.3.3 adds the possibility of having non-unitary parameters, while Section 2.3.4 generalizes the game to directed graphs. Finally, Section 2.3.5 investigates structural properties of special directed graph classes that can be explored to make at least PROTECTION polynomially solvable.

	Undirected Graphs		Directed Graphs	
Decision Versions	Unitary Case *Section 2.3.2*	Weighted Case *Section 2.3.3*	Unitary Case *Sections 2.3.4 & 2.3.5*	Weighted Case
Protect	[1] NP-complete	[6] NP-complete	[11] NP-complete	[16] NP-complete
Attack	[2] Polynomial	[7] NP-complete	[12] NP-complete	[17] NP-complete
Attack-Protect	[3] NP-hard	[8] Σ_2^p-complete	[13] NP-hard	[18] Σ_2^p-complete
Vaccination-Attack	[4] NP-complete	[9] Σ_2^p-complete	[14] Σ_2^p-complete	[19] Σ_2^p-complete
MCN	[5] NP-hard	[10] Σ_3^p-complete	[15] Σ_2^p-hard	[20] Σ_3^p-complete

Table 2.2. Computational complexity of the decision versions of the subproblems in MCN. Entries in gray correspond to results that follow as corollaries. In increasing order, we have: [1] $\Longrightarrow$ [6], [4] $\Longrightarrow$ [5], [12] $\Longrightarrow$ [13], [14] $\Longrightarrow$ [15], and [6-10] $\Longrightarrow$ [16-20].

2.3.1. Complexity of related problems

One way to measure the robustness of a given network is to study its connectivity properties, for which many metrics exist. With respect to a fixed metric, nodes often play different roles in the graph, with varying levels of importance. The most important nodes are qualified as *critical*. Thus, the problem of detecting subsets of critical nodes with respect to some connectivity measure is of great interest.

Critical Node Detection Problems (CNDP). The CNDPs have been extensively studied, with names varying with the connectivity metric to optimize and the constraints of the problem. Many of its studied versions have been shown to be NP-complete on general graphs; see [Lalou et al., 2018] for a recent survey. Indeed, many of these belong to the class of problems called *Node-Deletion Problems* [Lalou et al., 2018]. They consist in deleting the smallest subset of nodes from a graph so that the induced subgraph satisfies a certain property Π. Lewis and Yannakakis 1980 have shown that if Π is *nontrivial* and *hereditary*, then the subsequent node deletion problem is NP-hard. In particular, MinMaxC, the problem of finding a set of nodes D from a graph G with a budget constraint $|D| \leq \Omega$ such that the removal of D minimizes the size of the largest connected component in the remaining graph, has been shown to be NP-hard in the strong sense thanks to this argument [Shen et al., 2012]. Moreover, some CNDP problems remain NP-hard even on particular graph classes [Addis et al., 2013; Lalou et al., 2018]. For example, the original *Critical Node Problem* (CNP) [Arulselvan et al., 2009] which seeks to minimize the *pairwise connectivity* of the graph by removing a limited number of nodes remains NP-hard on split or bipartite graphs [Addis et al., 2013].

Interdiction Games. In several CNDP, although the optimization problem is formulated with a natural single objective, the task is inherently constituted of several ones. In the

CNP, minimizing the pairwise connectivity maximizes the number of connected components in the residual graph, while simultaneously minimizing the variance in the component sizes [Arulselvan et al., 2009]. Even though in this particular case, it has been shown that the multi-objective formulation is not equivalent to the original one [Ventresca et al., 2018], splitting the objective in two is sometimes possible. For example, Furini et al. 2019a exhibited the hidden bilevel structure of the *Capacitated Vertex Separator problem* by formulating it as a two player Stackelberg game in which a leader interdicts the network by removing some of its nodes and a follower determines the maximum connected component in the remaining graph, highlighting the link between CNDP problems and *Interdiction Games*. Interdiction games on networks are a special family of two-player zero-sum Stackelberg games in which a leader interdicts parts of the network *(arcs or nodes)* subject to a budget limitation in order to maximize the disruption of the follower's objective who solves an optimization problem on the remaining graph *(*e.g.*, the maximum flow or the maximum clique)*. Whereas some interdiction games such as the *network flow interdiction* are NP-complete [Wood, 1993], others such as the *binary knapsack interdiction problem* [DeNegre, 2011; Caprara et al., 2014] or the *maximum clique interdiction game* [Furini et al., 2019b] have been shown to be Σ_2^p-complete, shading light on the intrinsic relationship between this class of problems and the second level of the polynomial hierarchy.

However, the unitary undirected version of MCN, as originally introduced by Baggio et al. 2020, is not an interdiction problem per se but *contains* one. Indeed, the vaccination stage of the game focuses on identifying critical infrastructures in the network to interdict them preventively to counter an intentional attack, which falls into the framework of Network Interdiction problems. Nevertheless, the game does not finish with the attack: there is a third stage where the defender tries to isolate the propagation of the infection to maximize the unharmed fraction of the network. Finding a blocking strategy to limit the diffusion of an infection is related to the *Firefighter problem*, which has been shown to be NP-complete, even for trees of maximum degree three [Finbow et al., 2007]. Thus, the MCN problem combines two different paradigms in network protection, *prevention* and *blocking*, each being related to provably hard problems. The overall contraction leads to a trilevel optimization formulation for the MCN, making it fall under the *Defender-Attacker-Defender* (DAD) framework introduced by Brown et al. 2006 to study the defense of critical infrastructure against malicious attacks.

Defender-Attacker-Defender. Although the general DAD has been claimed to be NP-hard in [Martin, 2007], complexity results for trilevel combinatorial optimization problems are scarce. In [Johannes, 2011], a new proof that *Trilevel Linear Programming* is Σ_2^p-hard is provided, building upon the results in [Blair, 1992; Dudás et al., 1998; Jeroslow, 1985] showing that the *Multilevel Linear Programming* problem with $L+1$ levels is Σ_L^p-hard. In fact, the decision version of MCN problem can be formulated as *"given* 3 *integer budgets* Ω, Φ, Λ*, a graph* G *and an integer* K*, is there a vaccination* D *such that for all attacks* I *there exists a protection* P *saving at least* K *nodes?"* Thus, there seems to be a link between the MCN and the 3-*alternating quantified satisfiability* problem which has been shown to be Σ_3^p-complete by Meyer, Stockmeyer and Wrathall [Meyer and Stockmeyer, 1972; Wrathall, 1976], making one expect the MCN to be complete for this class.

2.3.2. Undirected graphs: the unitary case

In this section, we focus on undirected graphs $G = (V, E)$, *i.e.*, for each couples of nodes $(u,v) \in V \times V$, if the arc (u,v) is in G, then (v,u) is also in the graph. We thus call E the set of undirected edges. Here, we also consider unit benefits and costs, *i.e.*, $\forall v \in V,\ \hat{c}_v = h_v = c_v = b_v = 1$. We introduce s, the function that, given a graph G, the vaccination strategy D, the attack strategy I and the protection strategy P, returns $s(G, D, I, P)$, the number of saved nodes in the end of the game. Thus, in this setting, the trilevel formulation of the problem is simply:

$$\max_{\substack{D\subseteq V\\|D|\leq\Omega}} \min_{\substack{I\subseteq V\backslash D\\|I|\leq\Phi}} \max_{\substack{P\subseteq V\backslash(I\cup D)\\|P|\leq\Lambda}} s(G, D, I, P). \tag{2.3.1}$$

To ease our analysis, guided by the relationship between *Critical Node Detection Problems* and *Node-Deletion Problems*, we first write the immediate Property 2.3.1 stating that *vaccinating* or *protecting* nodes has almost the same effect as *removing* them from the graph with respect to s. Starting from $G = (V, E)$ and a subset $W \subseteq V$, we denote by $G[V\backslash W]$ the graph resulting from the deletion of the nodes in W and its incident edges.

Property 2.3.1. *Given* G, D, I, P*, we have that* $s(G, D, I, P) = s(G[V\backslash(D \cup P)], \emptyset, I, \emptyset) + |D| + |P|$

What Property 2.3.1 actually says is that the infected nodes in G are the ones in the connected components of $G[V\backslash(D \cup P)]$ where there is at least one attacked node in I.

We will start by classifying the computational complexity of Protect, followed by the one of Attack-Protect, and, finally, Vaccination-Attack. From the latter, we obtain the complexity of Attack, and the minimum complexity of MCN.

2.3.2.1. The Protection problem

In Protect, the defender is given D and I and seeks to find an optimal P. Thus, thanks to Property 2.3.1, we can assume that the game takes place in $G_a = G[V\backslash D]$ for this last move: the defender wants to find at most Λ nodes $P \subseteq V_a \backslash I$ that will maximixe $s(G_a, \emptyset, I, P)$. For a given choice of P, we introduce $C_1(P), ..., C_{N(P)}(P)$, the $N(P)$ connected components in the graph $G_a[V_a \backslash P]$. Hence, the objective of the defender being to find P minimizing the number of infected nodes $f(P)$, we can define it as:

$$f(P) = \sum_{i=1}^{N(P)} |C_i(P)| \times \mathbb{1}_{C_i(P) \cap I \neq \emptyset}. \tag{2.3.2}$$

We will show that finding such a P is NP-complete. We argue that it is a direct consequence of the results of [Addis et al., 2013] showing that the *Critical Node Problem* is NP-hard on split graphs.

The Critical Node Problem on split graphs. The *Critical Node Problem (CNP)* [Arulselvan et al., 2009] is a related problem to ours. The setting is very similar to PROTECTION: we have an undirected graph $\bar{G} = (\bar{V}, \bar{E})$, an integer budget B, and we want to find a subset $\bar{P}$ of nodes to remove that minimizes the *pairwise connectivity* of the residual subgraph $\bar{G}[\bar{V} \backslash \bar{P}]$ under the constraint of having $|\bar{P}| \leq B$. If we denote by $\bar{C}_1(\bar{P}), ..., \bar{C}_{N(\bar{P})}(\bar{P})$ the $N(\bar{P})$ connected components of $\bar{G}[\bar{V} \backslash \bar{P}]$, the measure we want to minimize is:

$$g(\bar{P}) = \sum_{i=1}^{N(\bar{P})} \binom{|\bar{C}_i(\bar{P})|}{2} \tag{2.3.3}$$

where each term in the sum is the pairwise connectivity of $\bar{C}_i(\bar{P})$. Here, we will focus more particularly on *split graphs*. A split graph is a graph $\bar{G} = (\bar{V}, \bar{E})$ whose nodes $\bar{V}$ can be split in two sets $\bar{V}_1$ and $\bar{V}_2$, $\bar{V}_1$ forming a clique and $\bar{V}_2$ an independent set. Thus, the decision problem for this particular case of the CNP is:

CNP$_{split}$:
INSTANCE: A split graph $\bar{G} = (\bar{V}_1, \bar{V}_2; \bar{E})$, a non-negative integer budget $B \leq |\bar{V}|$ and a non-negative integer $\bar{K}$.
QUESTION: Is there a subset $\bar{P} \subseteq \bar{V}$, $\bar{P} \leq B$ such that $g(\bar{P}) \leq \bar{K}$?

As Addis et al. 2013 noted, in this setting there is at most one connected component of the residual subgraph $\bar{G}[\bar{V} \backslash \bar{P}]$ that contains more than one node. Moreover, it is easy to see that if this *nontrivial connected component* exists, it necessarily contains a subclique of $\bar{G}[\bar{V}_1]$. More than that, it is the only connected component of $\bar{G}[\bar{V} \backslash \bar{P}]$ containing nodes from $\bar{V}_1$. Thus, we can name $\bar{C}_1$ the connected component containing nodes of $\bar{V}_1$ *(in the case of $\bar{P} \supseteq \bar{V}_1$, then $\bar{C}_1$ is either a singleton from $\bar{V}_2$ or is empty and our reasoning still holds)*. Then, minimizing (2.3.3) is equivalent to minimize $|\bar{C}_1|$. But finding the subset of nodes $\bar{P}$ to remove to do that has been shown to be NP-hard:

Lemma 2.3.2. *[Addis et al., 2013]* CNP$_{split}$ *is NP-hard.*

Complexity result. Next, we show that the decison version of PROTECT is NP-complete using a reduction from MCN$_{split}$. The decision problem is the following:

PROTECT:
INSTANCE: A graph $G_a = (V_a,E_a)$, a set of attacked nodes $I \subseteq V_a$, a non-negative integer budget $\Lambda \leq |V_a| - |I|$ and a non-negative integer K.
QUESTION: Is there a subset $P \subseteq V_a \backslash I$, $|P| \leq \Lambda$ such that the number of infected nodes $f(P) \leq K$?

Note that the question can be equivalently asked with the inequality $s(G_a, \emptyset, I, P) \geq |V_a| - K$.

Theorem 2.3.3. PROTECT *is NP-complete.*

PROOF. It is easy to see that PROTECT is NP as determining the objective value only requires finding the connected components of $G_a[V_a \backslash P]$ which can be done in linear time using a depth-first search (DFS).
To complete the proof, we exhibit an immediate reduction from CNP$_{split}$. Let us take an instance of this problem, *i.e.* a split graph $\bar{G} = (\bar{V}_1, \bar{V}_2; \bar{E})$, a non-negative integer budget B and a non-negative integer $\bar{K}$. Given that, we build a graph G_a by growing by one the size of the clique $\bar{G}[\bar{V}_1]$ with the addition of a node u. Thus, $V_a = \bar{V}_1 \cup \{u\} \cup \bar{V}_2$ and E_a is obtained by taking $\bar{E}$ and adding an edge $(u,\bar{v}_1)$ $\forall \bar{v}_1 \in \bar{V}_1$. In fact, the new graph is still a split graph $G_a = (\bar{V}_1 \cup \{u\}, \bar{V}_2; E_a)$. Finally, the corresponding instance of PROTECT is given by G_a, $I = \{u\}$, $\Lambda = B$ and $K = \left\lfloor \frac{1}{2}(3 + \sqrt{8\bar{K} + 1}) \right\rfloor$ (obtained by solving $\bar{K} = \binom{K-1}{2}$). An example of such construction can be found in Figure 2.5. Then, as there is only one attacked node, minimizing (2.3.2) on this instance of PROTECT corresponds to chose a P that minimizes the size of the unique connected component to which u belongs in $G_a[V_a \backslash P]$. Let's name C_1 this connected component. But as u belongs to the clique part of the split graph G_a, C_1 is also the unique connected component of $G_a[V_a \backslash P]$ containing nodes from $V_1 = \bar{V}_1 \cup \{u\}$. Thus, we have that $C_1 = \bar{C}_1 \cup \{u\}$ and $g(P) = \begin{pmatrix} f(P) - 1 \\ 2 \end{pmatrix}$. Hence, finding P that minimizes f on G_a is equivalent to finding P that minimizes g on $\bar{G}$. This finishes the proof that PROTECT is NP-hard. $\square$

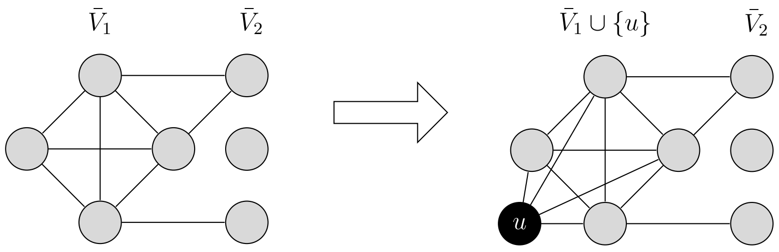


Figure 2.5. Graph reduction from CNP$_{split}$ to Protect

2.3.2.2. The Attack-Protect problem

We showed that solving the last level of MCN is NP-complete, now we will prove that Attack-Protect is also NP-hard. In this bilevel problem, we are taking the side of the attacker: the aim is to find the attack that will maximize the number of infected nodes after protection. The decision version of the problem is:

Attack-Protect:
instance: A graph $G_a = (V_a, E_a)$, two non-negative integer budgets Φ, Λ such that $\Phi + \Lambda \leq |V_a|$ and a non-negative integer $K \leq |V_a|$
question: Is there a subset $I \subseteq V_a$, $|I| = \Phi$ such that $\forall P \subseteq V_a \backslash I$, $|P| \leq \Lambda$, the number of infected nodes $f(P) \geq K$?

We will use a reduction from the *Dominating Set* problem, a known NP-complete problem [Garey and Johnson, 1979], whose decision version is:

Dominating Set:
instance: A graph $\bar{G} = (\bar{V}, \bar{E})$, a positive integer $B \leq |\bar{V}|$
question: Is there a subset $U \subseteq \bar{V}$, $|U| \leq B$, such that $\forall v \in \bar{V} \backslash U$, $\exists\ u \in U$ such that $(u,v) \in \bar{E}$?

Theorem 2.3.4. Attack-Protect *is NP-hard.*

Proof. Let us take a graph $\bar{G} = (\bar{V}, \bar{E})$ and a positive integer $B \leq |\bar{V}|$. The instance of Attack-Protect is simply created by taking $G_a = \bar{G}$, $\Phi = B$, $\Lambda = |V_a| - \Phi - 1$ and $K = \Phi + 1$. In this configuration, we have a protection budget Λ which is exactly one less than the number of nodes that are not attacked. Thus, if all the protection budget is spent, there is only one node u in the graph that is neither attacked nor protected. Therefore, if u becomes infected after protection *(i.e $f(P) = K = \Phi + 1$)*, that means that the protection strategy did not manage to save one unit of budget while saving all the other nodes, meaning that the other nodes were all in direct contact with at least one attacked one *(if it was not the case, one unit of budget could have been saved by protecting all the neighbors of the node that is not in direct contact with I)*. As u also becomes infected, it also means that it is adjacent to one node in I. Thus, finding I such that $\forall P, f(P) \geq K$ means that I is a dominating set of size B, which concludes the proof. □

2.3.2.3. The Vaccination-Attack problem

In this part, we will ignore the fact that there is a protection stage at the end. This is a particular case of MCN since it is equivalent to studying it with protection budget $\Lambda = 0$. We will show that the bilevel problem Vaccination-Attack is NP-complete. The decision problem is the following:

Vaccination-Attack:
instance: A graph $G = (V,E)$, two non-negative integer budgets Ω and Φ such that $\Omega + \Phi \leq |V|$ and a non-negative integer K.
question: Is there a subset $D \subseteq V$, $|D| \leq \Omega$ such that $\forall I \subseteq V \backslash D$ with $|I| \leq \Phi$, the number of infected nodes $|V| - s(G, D, I, \emptyset) \leq K$?

First, we argue that in this configuration, finding the optimal attack following a given vaccination can be done in polynomial time.

Lemma 2.3.5. Vaccination-Attack $\in$ *NP. Moreover,* Attack *can be solved in polynomial time.*

Proof. Given a vaccinated set D, we want to verify that all the possible subsequent attacks cannot infect more than K nodes. To do that, it suffices to find the best attack, *i.e.*, solve the Attacker optimization problem, and check whether or not it complies with the inequality. But, as we highlighted it with Property 2.3.1, the graph on which the attack phase takes place is $G_a = G[V \backslash D]$ and the saved nodes in the end are exactly the ones in the connected components of G_a that do not contain any attacked node. Thus, the best attack possible given G_a and budget Φ is to infect one node in each of the Φ largest connected components of G_a. This can be done in linear time using a DFS. Hence, Vaccination-Attack $\in$ NP. □

In fact, this proof showed that Vaccination-Attack is actually equivalent to another problem: finding a subset of nodes D to remove from G that minimizes the sum of the sizes of the Φ largest connected components in the induced subgraph. Let's call this problem MinMaxΦC:

MinMaxΦC:
INSTANCE: A graph $G = (V,E)$, two non-negative integer budgets Ω and Φ such that $\Omega + \Phi \leq |V|$ and a non-negative integer K.
QUESTION: Is there a subset $D \subseteq V$, $|D| \leq \Omega$ such that the sum of the sizes of the Φ largest connected component in $G[V \backslash D]$ is less than K ?

Lemma 2.3.6. Vaccination-Attack *and MinMaxΦC are equivalent problems.*

Shen et al. 2012 argued that MinMax1C, the problem that only seeks to minimize the size of the largest connected component, is NP-hard as a direct consequence of the **Theorem 1** of Yannakakis 1978. Indeed, this theorem states that:

Theorem 2.3.7. *[Yannakakis, 1978] The node-deletion problem for nontrivial, interesting graph-properties Π that are hereditary on induced subgraphs is NP-hard. If Π is easy to recognize (i.e in P), then the problem is NP-complete.*

Here, we propose to properly show that our more general problem MinMaxΦC is NP-complete thanks to Theorem 2.3.7. To do that, we first recall the definitions of nontrivial, interesting and hereditary graph properties. Then, we exhibit the property Π we are interested in here and show that it falls under the scope of the theorem.

Corollary 2.3.8. Vaccination-Attack *is NP-complete.*

Proof. We use Theorem 2.3.7. We recall the definitions of nontrivial, interesting and hereditary properties as written in [Yannakakis, 1978]:

— Π is **nontrivial** if it is true for a single node and is not satisfied by all the graphs in a given input domain.
— Π is **interesting** if there are arbitrarily large graphs satisfying it.
— Π is **hereditary** if the fact that it is satisfied on G implies that $\forall v \in V$, it is satisfied on $G[V \backslash \{v\}]$.

In MinMaxΦC, we define the following property Π on a graph G with both K and Φ greater or equal to one: *the sum of the sizes of the Φ largest connected component of G is less than K.* It is easy to see that Π is nontrivial as it is satisfied for a single node and there are graphs that violate Π, it is interesting as we can build arbitrarily large graphs satisfying Π (e.g by adding arbitrarily many small connected components to graphs satisfying Π with already strictly more than Φ connected components), and is obviously hereditary. Thus, Theorem 2.3.7 tells us that MinMaxΦC is NP-hard. Π being easy to check in linear time, MinMaxΦC is NP-complete, and by Lemma 2.3.6, so is Vaccination-Attack. □

Corollary 2.3.9. MCN *is NP-hard.*

Proof. Given an instance of Vaccination-Attack, there is a corresponding instance of MCN by taking the same G, Ω, Φ, K and by setting $\Lambda = 0$. □

2.3.3. Undirected graphs: the weighted case

In this section, we study the version of MCN presented in problem (1) restricted to undirected graphs. We will use the subscript w to denote the weighted version, MCN$_w$, as well as its subgames. In this problem, given a graph $G = (V,E)$, each node $v \in V$ is associated with a benefit b_v and cost parameters $\hat{c}_v, h_v$ and c_v, respectively the cost of vaccinating, attacking and protecting node v. Having introduced costs and benefits, this problem is thus intimately related to *Knapsack problems*, which we will use to demonstrate all of our complexity results in this part. First, we will highlight the direct relationship between Attack$_w$ and Knapsack, which will get us the NP-completeness of this problem. Then, we will focus on the two bilevel sub-problems Vaccination-Attack$_w$ and Attack-Protect$_w$ and prove they are Σ_2^p-complete thanks to a *Knapsack Interdiction problem.* Finally, we show that a trilevel Knapsack interdiction is Σ_3^p-complete, and use this result to prove that MCN$_w$ is Σ_3^p-complete.

Note that the NP-completeness of Protect$_w$ is immediate from the previous section. Hence, we start with complexity results to Attack$_w$, then Attack-Protect$_w$, and Vaccination-Attack$_w$, and, to conclude, MCN$_w$. We will observe that the introduction of non-unitary parameters offers sufficient flexibility to go a level up in the polynomial hierarchy in comparison with the unitary undirected cases.

2.3.3.1. The Attack$_w$ problem

In the attack phase, the vaccination already took place so we effectively work on G_a, which is the result of the deletion of the vaccinated nodes from the original graph. We are given a non-negative attack budget Φ, and as there is no protection phase afterwards, we set $\Lambda = 0$. The goal is thus to harvest the most benefit possible by infecting nodes subject to a budget limit. The decision version of the problem is then:

Attack$_w$:
instance: An undirected graph $G_a = (V_a, E_a)$, a non-negative integer cost h_v and value b_v for each node $v \in V$, a non-negative integer budget Φ, and a non-negative integer K.
question: Is there a subset of nodes $I \subseteq V_a$ to attack, with cost $\sum_{v \in I} h_v \leq \Phi$ such that

the sum of the benefits of the resulting infected nodes in G_a is greater or equal to K?

To make evident the NP-completeness of the problem, we simply state the decision version of the *Knapsack problem*, one of the Karp's 21 NP-complete problems [Karp, 1972]:

KNAPSACK:
INSTANCE: Finite set U, for each $u \in U$, a positive integer size $a_u \in \mathbb{N}$ and a positive integer profit $p_u \in \mathbb{N}$, and two positive integers B and $\bar{K}$.
QUESTION: Is there a subset $U' \subseteq U$ such that $\sum_{u \in U'} a_u \leq B$ verifying $\sum_{u \in U'} p_u \geq \bar{K}$?

Theorem 2.3.10. *ATTACK$_w$ is NP-complete, even on trivial graphs.*

PROOF. It is easy to see that ATTACK$_w$ $\in$ NP. Indeed, given an attack I, finding the subsequent infected nodes can be done in linear time thanks to a DFS. Then, it suffices to sum the values associated with each infected nodes to verify that it is greater or equal to K.
The reduction from KNAPSACK we use to show the NP-hardness is straightforward. Given an instance of KNAPSACK, we set $V_a = U$, $E_a = \emptyset$, $K = \bar{K}$, $\Phi = B$, and $\forall v \in V_a$, $h_v = a_v$, $b_v = p_v$. In this configuration, G_a having no edges, the attacked nodes are exactly the infected ones in the end, and the goal of the attacker is equivalent to filling up a knapsack with limited capacity by choosing which nodes to attack. □

2.3.3.2. The ATTACK-PROTECT$_w$ problem

In the proof of Theorem 2.3.10, we highlighted how a KNAPSACK instance can be directly transformed into a weighted graph with no edges. In this section, as well as in the one that follows, we will use a similar transformation, but add one additional *root node* to our construction in order to build a star graph: one root node connected with an edge to each of the other nodes, each one representing an item of the knapsack. That way, the complexity results we devise also hold for trees.

As before, the vaccination having already been done, we start from G_a, the graph where the vaccinated nodes have been removed.

ATTACK-PROTECT$_w$:
INSTANCE: A graph $G_a = (V_a, E_a)$, a non-negative integer K, two non-negative integer budgets Φ and Λ, $\forall v \in V_a$ two non-negative integer costs h_v, c_v and a non-negative integer benefit b_v.
QUESTION: Is there a subset $I \subseteq V_a$, with cost $\sum_{v \in I} h_v \leq \Phi$ such that $\forall P \subseteq V_a \backslash I$ with cost

$\sum_{v \in P} c_v \leq \Lambda$, the sum of the benefit of the saved nodes is strictly less than K?

In order to show that ATTACK-PROTECT$_w$ is Σ_2^p-complete, we use the *Bilevel Interdiction Knapsack Problem* introduced by DeNegre [DeNegre, 2011] and proven to be Σ_2^p-complete in [Caprara et al., 2014]. In this problem, two players, a leader and a follower, can select items in the same set of objects O. First, the leader packs some items into her knapsack, then the follower chooses among the remaining ones. The aim of the leader is to interdict a subset of items, subject to a capacity constraint, in order to minimize the total profit of the follower. The objective of the follower is to maximize its profit, subject to a constraint capping the maximum profit obtainable for her. The decision problem is then:

BILEVEL INTERDICTION KNAPSACK (BIK):
INSTANCE: A set of items O such that each $o \in O$ has a positive integer weight a_o and a positive integer profit p_o, a positive integer maximum weight capacity A for the leader, a positive integer maximum profit B for the follower, and a positive integer $\bar{K} \leq B$.
QUESTION: Is there a subset $O_l \subseteq O$ of items for the leader to select, with $\sum_{o \in O_l} a_o \leq A$, such that every subset $O_f \subseteq O \setminus O_l$ with $\sum_{o \in O_f} p_o \leq B$ that the follower can create has a total profit $\sum_{o \in O_f} p_o < \bar{K}$?

Theorem 2.3.11. ATTACK-PROTECT$_w$ *is strongly* Σ_2^p*-complete, even if the graph is a tree.*

PROOF. First, ATTACK-PROTECT$_w$ is in Σ_2^p since this decision problem is exactly of the form $\exists I \ \ \forall P \ \ Q(I,P)$.
Next, we prove the problem Σ_2^p-hardness. Let us begin by noting that we can restrict the instances of KIP to the ones where $\bar{K}$ and B are strictly inferior to $\sum_{o \in O} p_o$, otherwise, KIP is trivial to solve. This remark is used in the second part of this proof.
Starting from an instance of BIK, we construct an instance of ATTACK-PROTECT$_w$ as follows. We first build a star graph $G_a = (V_a, E_a)$ with a root node r and a node v_o for each $o \in O$ linked to r through an edge (r, v_o). We set $b_r = \sum_{o \in O} p_o + 1$ and $h_r = c_r = 1$. We also set $b_{v_o} = c_{v_o} = p_o$ and $h_{v_o} = a_o$ for each $o \in O$. See Figure 2.6. Finally, we set $\Phi = A + 1$, $\Lambda = B$ and $K = \bar{K}$.

Suppose first that BIK is a *Yes* instance. Then, there is a set of items $O_l \subseteq O$ of total weight $\sum_{o \in O_l} a_o \leq A$ such that for all $O_f \subseteq O \setminus O_l$ feasible for the follower, it holds $\sum_{o \in O_f} p_o \leq \bar{K} - 1$. Consequently, in the ATTACK-PROTECT$_w$, the attacker can select the subset of nodes $I = \{r\} \cup \{v_o : \ o \in O_l\}$ with a feasible attacking cost $\sum_{v \in I} h_v = 1 + \sum_{o \in O_l} a_o \leq A + 1 = \Phi$. Now, the defender can only protect nodes in $\{v_o : \ o \notin O_l\}$ and since the central node of the star graph is infected, the saved nodes will

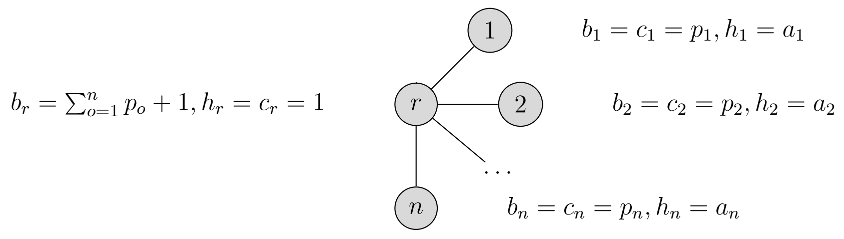

$b_1 = c_1 = p_1, h_1 = a_1$
$b_r = \sum_{o=1}^{n} p_o + 1, h_r = c_r = 1$
$b_2 = c_2 = p_2, h_2 = a_2$
$b_n = c_n = p_n, h_n = a_n$
1
r
2
n
...

Figure 2.6. Graph reduction from BIK to Attack-Protect$_w$ when $O = \{1,2\ldots,n\}$.

be the protected ones. The aim of the defender is therefore to select the subset of nodes of maximum total benefit with respect to the protection budget Λ. This is exactly the follower's problem in BIK. Hence, since BIK is an *Yes* instance, the defender (follower in BIK) cannot attain a benefit (profit in BIK) equal or greater to $K = \bar{K}$ through a feasible action. Therefore, the Attack-Protect$_w$ is a *Yes* instance.

Now suppose that Attack-Protection$_w$ is a *Yes* instance. Thus, there exists an attack strategy $I \subseteq V_a$ such that there is **no** feasible subset $P \subseteq V_a \setminus I$ of protected nodes leading to a total benefit greater or equal to K for the defender. As $\Phi \geq 1$, it is obvious that the attacker will attack at least the central node r, otherwise, the defender would pick it and achieve a benefit superior to K (recall that $K = \bar{K} < \sum_{o \in O} p_o$), contradicting Attack-Protection$_w$ *Yes* instance. Hence, the attacker is left with budget $\Phi - h_r = A$. Once the central node is attacked, only the other nodes subsequently protected will not be infected. Therefore, the rest of the attack budget A is spent on a subset of nodes of $\{v_o \in V_a : \ o \in O\}$ and it ensures that for any $P = \{v_o \in V_a : \ o \in O \setminus I\}$ with $\sum_{v \in P} c_v = \sum_{o:v_o \in P} p_v \leq \Lambda = B$, the total benefit for the defender is $\sum_{v \in P} b_v = \sum_{o:v_o \in P} p_v \leq \bar{K} - 1$. Consequently, BIK is also a *Yes* instance.

This completes the proof that Attack-Protect$_w$ is Σ_2^p-complete. Moreover, since the BIK was shown to be NP-complete even for unary encoding, we can conclude that no pseudopolynomial-time algorithm exists to solve the Attack-Protect subgame. Since a star graph is a tree, the result stated in the theorem holds. □

2.3.3.3. The Vaccination-Attack$_w$ problem

Using a similar reduction to the one in the proof of Theorem 2.3.11, we show that the Vaccination-Attack$_w$ on weighted graphs is Σ_2^p-complete. As in the unitary case, this is equivalent to studying MCN$_w$ problems where we set $\Lambda = 0$. The decision version of the problem is:

VACCINATION-ATTACK$_w$:
INSTANCE: A graph $G = (V,E)$, a non-negative integer K, two non-negative integer budgets Ω and Φ, $\forall v \in V$ two non-negative integer costs $\hat{c}_v$, h_v and a non-negative integer benefit b_v.
QUESTION: Is there a subset $D \subseteq V$, with cost $\sum_{v \in D} \hat{c}_v \leq \Omega$ such that $\forall I \subseteq V \setminus D$ with cost $\sum_{v \in I} h_v \leq \Phi$, the sum of the benefit of the infected nodes is strictly less than K?

Theorem 2.3.12. VACCINATION-ATTACK$_w$ *is strongly* Σ_2^p*-complete, even if the graph is a tree.*

PROOF. As before, VACCINATION-ATTACK$_w$ is in Σ_2^p since this decision problem is exactly of the form $\exists D \;\; \forall I \;\; Q(D,I)$.
Now, we establish the problem Σ_2^p-hardness. We start from an instance of BIK, defined in the previous section, and we then construct an instance of VACCINATION-ATTACK$_w$ as follows. First, we build a star graph $G = (V,E)$ with a central node r and $|O|$ leaf nodes v_o with $o \in O$. See Figure 2.7. We add an edge (r,v_o) for each such leaf node. The central node has benefit $b_r = \bar{K}$ and costs $\hat{c}_r = h_r = 1$. Each leaf node v_o with $o \in O$ has a benefit $b_{v_o} = p_o$, cost for the defender $\hat{c}_{v_o} = a_o$ and cost for the attacker $h_{v_o} = p_o$. Finally, we fix $\Omega = A + 1$, $\Phi = B$ and $K = \bar{K}$.

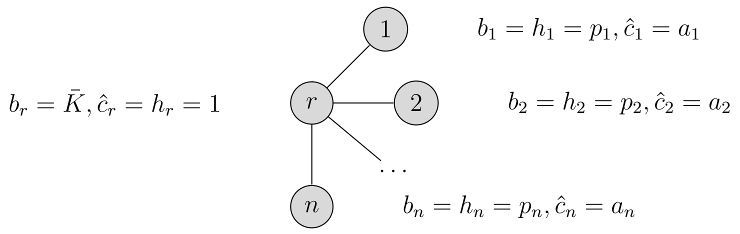

$b_1 = h_1 = p_1, \hat{c}_1 = a_1$
$b_r = \bar{K}, \hat{c}_r = h_r = 1$
r
1
2
$b_2 = h_2 = p_2, \hat{c}_2 = a_2$
...
n
$b_n = h_n = p_n, \hat{c}_n = a_n$

Figure 2.7. Graph reduction from BIK to VACCINATION-ATTACK$_w$ when $O = \{1,2\ldots,n\}$.

This is exactly the setting of BIK and one can easily complete the proof of equivalence of the two decision instances following a path very similar to the proof of Theorem 2.3.11. Finally, the reduction used a star graph which is a particular case of a tree. Hence, the problem is Σ_2^p-complete even on trees. □

2.3.3.4. The MCN$_w$ problem

In this section we show that the decision problem MCN$_w$ is Σ_3^p-complete.

MCN$_w$:
INSTANCE: A graph $G = (V,E)$, a non-negative integer K, three non-negative integer budgets Ω, Φ and Λ, $\forall v \in V$ three non-negative integer costs $\hat{c}_v$, h_v and c_v, and a non-negative integer benefit b_v.
QUESTION: Is there a subset $D \subseteq V$, with cost $\sum_{v \in D} \hat{c}_v \leq \Omega$ such that $\forall I \subseteq V \setminus D$ with cost

$\sum_{v \in I} h_v \leq \Phi$, there is $P \subseteq V \backslash I$ with cost $\sum_{v \in D} c_v \leq \Lambda$ such that the sum of the benefit of the saved nodes is greater or equal to K?

In other to achieve our ultimate goal, we take the *3-Alternating Quantified Satisfiability* problem ($B_3 \cap 3CNF$), known to be Σ_3^p-complete problem [Stockmeyer and Meyer, 1973; Wrathall, 1976], in order to prove that the generalization of BIK to a trilevel, the *Trilevel Interdiction Knapsack* (TIK), is Σ_3^p-complete. Then, TIK is used to demonstrate that MCN$_w$ is Σ_3^p-complete.

3-Alternating Quantified Satisfiability ($B_3 \cap 3CNF$):
instance: Disjoint non-empty sets of variables X, Y and Z, and a Boolean expression E over $U = X \cup Y \cup Z$ in conjunctive normal form with at most 3 literal in each clause $c \in C$.
question: Is there a 0-1 assignment for X so that for all 0-1 assignments of Y there is a 0-1 assignment of Z such that E is satisfied?

Trilevel Interdiction Knapsack (TIK):
instance: A set of items O such that each $o \in O$ has two a positive integer weights a'_o and a_o and a positive integer profit p_o, two positive integer maximum weight capacities A' and A, a positive integer maximum profit B and a positive integer goal $\bar{K} \leq B$.
question: Is there a subset $O_1 \subseteq O$ of items, with $\sum_{o \in O_1} a'_o \leq A'$, such that every subset $O_2 \subseteq O \setminus O_1$, with $\sum_{o \in O_2} a_o \leq A$, there is a subset $O_3 \subseteq O \setminus O_2$, with $\sum_{o \in O_3} p_o \leq B$, such that $\sum_{o \in O_3} p_o \geq \bar{K}$ holds?

Theorem 2.3.13. *TIK is Σ_3^p-complete.*

Proof. The statement of TIK is of the form $\exists O_1 \quad \forall O_2 \quad \exists O_3 \quad Q(O_1,O_2,O_3)$, directly implying that it is in Σ_3^p.
Next, we use a reduction from the $B_3 \cap 3CNF$ which is very much in line with the reduction from 3-SAT to Subset Sum presented in [Cormen et al., 2009, Theorem 34.15]:

— For each variable $u \in U$, we create two items o_u and $o_{\bar{u}}$, one for each possible 0-1 assignment of u. We designate by $O_U = \{o_u : u \in U\}$ and $O_{\bar{U}} = \{o_{\bar{u}} : u \in U\}$ the two sets of items of size $|U|$.
— For each clause $c \in C$, *(i)* if c has 1 literal, we create one item o_c^1, *(ii)* if c has 2 literals, we create two items o_c^1 and o_c^2, and *(iii)* if c has 3 literals, we create three items o_c^1, o_c^2 and o_c^3. We designate by O_C the set of items associated with C.
— Weights, profits, maximum capacities, maximum profit and goal will be given by digits of size $|X| + |Y| + |Z| + |C| + 1$ in base 10. Hence, each digit position is labeled

by a variable or a clause: the first $|C|$ positions (least significant numbers) are labeled by the clauses, then the next $|X|$ positions are labeled by the variables X, then the next $|Y|$ positions are labeled by the variables Y, then the next $|Z|$ positions are labeled by the variables Z, and, finally, the last position is labeled as *forbidden*.

— For each $u \in U$, the two corresponding items o_u and $o_{\bar{u}}$ have weights and profits as described next. The weights and profits a'_{o_u}, a_{o_u}, p_{o_u}, $a'_{o_{\bar{u}}}$, $a_{o_{\bar{u}}}$ and $p_{o_{\bar{u}}}$ have digit 1 in the position labeled by the variable U and 0 in the positions labeled by other variables; the remaining digits are zero for a'_{o_u}, a_{o_u}, $a'_{o_{\bar{u}}}$ and $a_{o_{\bar{u}}}$. In particular, for all $o \in O_U \cup O_{\bar{U}}$, it holds $a'_{o_u} = a_{o_u}$ and $a'_{o_{\bar{u}}} = a_{o_{\bar{u}}}$.
If the literal u appears in clause $c \in C$, then p_{o_u} has digit 1 in the position labeled as c, and 0 otherwise. Similarly, if the literal $\neg u$ appears in clause $c \in C$, $p_{o_{\bar{u}}}$ has digit 1 in the position labeled by c, and 0 otherwise. Finally, for all $o \in O_U \cup O_{\bar{U}}$, p_{o_u} and $p_{o_{\bar{u}}}$ have digit 0 in the position labeled as forbidden.
— For each $c \in C$, the associated items have weights and profits as follows. If c has one literal, $a'_{o_c^1}$ and $a_{o_c^1}$ have 1 in the position labeled as forbidden and 0 elsewhere; $p_{o_c^1}$ has digit 3 in the position labeled as c and 0 elsewhere. If c has two literals, $a'_{o_c^1}$, $a'_{o_c^2}$, $a_{o_c^1}$ and $a_{o_c^2}$ have 1 in the position labeled as forbidden and 0 elsewhere; $p_{o_c^1}$ and $p_{o_c^2}$ have digit 3 and 2, respectively, in the position labeled as c and 0 elsewhere. If c has three literals, $a'_{o_c^1}$, $a'_{o_c^2}$, $a'_{o_c^3}$, $a_{o_c^1}$, $a_{o_c^2}$ and $a_{o_c^3}$ have 1 in the position labeled as forbidden and 0 elsewhere; $p_{o_c^1}$, $p_{o_c^2}$ and $p_{o_c^3}$ have digit 3, 2 and 1, respectively, in the position labeled as c and 0 elsewhere.
— The weight capacity A' has 1s for all digits with labels in X and 0s elsewhere. Hence, O_1 cannot contain items from $\{o_u, o_{\bar{u}} : u \in Z \cup Y\} \cup O_C$.
— The weight capacity A has 1s for all digits with labels in Y, 2s for all digits with labels in X and 0s elsewhere. Hence, O_2 cannot contain items from $\{o_u, o_{\bar{u}} : u \in Z\} \cup O_C$.
— The maximum profit B has 1s for all digits with labels in $X \cup Z$, 2s for all digits with labels in Y, 4s for all digits with labels in C, and 0s elsewhere. Hence, O_3 can take any item (as long as not interdicted by O_2).
— We make $\bar{K}$ is equal to B, except for the digits with labels Y, where it is 1.

See Figure 2.8 for an illustration of our reduction. Let $B_3 \cap 3CNF$ be a *Yes* instance. Then, take in O_1 the items o_u such that $u \in X$ is 1 and the items $o_{\bar{u}}$, otherwise. Clearly, this choice of O_1 respects the maximum weight A'. By construction, given this O_1, the best O_2 will take all items associated with X and not taken by O_1, as it does not interfere with the budget left for the items associated with Y. Furthermore, the optimal O_2 will also take

O			Z	Y	X		C		
		forbidden	d	c	b	a	c_1	c_2	c_3
o_a	$a'_{o_a} = a_{o_a}$	0	0	0	0	1	0	0	0
	p_{o_a}	0	0	0	0	1	1	0	1
$o_{\bar{a}}$	$a'_{o_{\bar{a}}} = a_{o_{\bar{a}}}$	0	0	0	0	1	0	0	0
	$p_{o_{\bar{a}}}$	0	0	0	0	1	0	1	1
o_b	$a'_{o_b} = a_{o_b}$	0	0	0	1	0	0	0	0
	p_{o_b}	0	0	0	1	0	1	0	1
$o_{\bar{b}}$	$a'_{o_{\bar{b}}} = a_{o_{\bar{b}}}$	0	0	0	1	0	0	0	0
	$p_{o_{\bar{b}}}$	0	0	0	1	0	0	1	0
o_c	$a_{o_c} = a'_{o_c}$	0	0	1	0	0	0	0	0
	p_{o_c}	0	0	1	0	0	0	0	1
$o_{\bar{c}}$	$a_{o_{\bar{c}}} = a'_{o_{\bar{c}}}$	0	0	1	0	0	0	0	0
	$p_{o_{\bar{c}}}$	0	0	1	0	0	1	0	1
o_d	$a'_{o_d} = a_{o_d}$	0	1	0	0	0	0	0	0
	p_{o_d}	0	1	0	0	0	0	1	0
$o_{\bar{d}}$	$a'_{o_{\bar{d}}} = a_{o_{\bar{d}}}$	0	1	0	0	0	0	0	0
	$p_{o_{\bar{d}}}$	0	1	0	0	0	0	0	0
$o^1_{c_1}$	$a'_{o^1_{c_1}}$	1	0	0	0	0	0	0	0
	$a_{o^1_{c_1}}$	1	0	0	0	0	0	0	0
	$p_{o^1_{c_1}}$	0	0	0	0	0	3	0	0
$o^2_{c_1}$	$a'_{o^2_{c_1}}$	1	0	0	0	0	0	0	0
	$a_{o^2_{c_1}}$	1	0	0	0	0	0	0	0
	$p_{o^2_{c_1}}$	0	0	0	0	0	2	0	0
$o^3_{c_1}$	$a'_{o^3_{c_1}}$	1	0	0	0	0	0	0	0
	$a_{o^3_{c_1}}$	1	0	0	0	0	0	0	0
	$p_{o^3_{c_1}}$	0	0	0	0	0	1	0	0
$o^1_{c_2}$	$a'_{o^1_{c_2}}$	1	0	0	0	0	0	0	0
	$a_{o^1_{c_2}}$	1	0	0	0	0	0	0	0
	$p_{o^1_{c_2}}$	0	0	0	0	0	0	3	0
$o^2_{c_2}$	$a'_{o^2_{c_2}}$	1	0	0	0	0	0	0	0
	$a_{o^2_{c_2}}$	1	0	0	0	0	0	0	0
	$p_{o^2_{c_2}}$	0	0	0	0	0	0	2	0
$o^3_{c_2}$	$a'_{o^3_{c_2}}$	1	0	0	0	0	0	0	0
	$a_{o^3_{c_2}}$	1	0	0	0	0	0	0	0
	$p_{o^3_{c_2}}$	0	0	0	0	0	0	1	0
$o^1_{c_3}$	$a'_{o^1_{c_3}}$	1	0	0	0	0	0	0	0
	$a_{o^1_{c_3}}$	1	0	0	0	0	0	0	0
	$p_{o^1_{c_3}}$	0	0	0	0	0	0	0	3
$o^2_{c_3}$	$a'_{o^2_{c_3}}$	1	0	0	0	0	0	0	0
	$a_{o^2_{c_3}}$	1	0	0	0	0	0	0	0
	$p_{o^2_{c_3}}$	0	0	0	0	0	0	0	2
$o^3_{c_3}$	$a'_{o^3_{c_3}}$	1	0	0	0	0	0	0	0
	$a_{o^3_{c_3}}$	1	0	0	0	0	0	0	0
	$p_{o^3_{c_3}}$	0	0	0	0	0	0	0	1
	A'	0	0	0	1	1	0	0	0
	A	0	0	1	2	2	0	0	0
	B	0	1	2	1	1	4	4	4
	$\bar{K}$	0	1	1	1	1	4	4	4

Figure 2.8. Example of construction of TIK from an instance $B_3 \cap 3CNF$ with $E = (a \vee b \vee \neg c) \wedge (\neg a \vee \neg b \vee d) \wedge (a \vee c \vee b)$, where $X = \{a, b\}$, $Y = \{c\}$, $Z = \{d\}$ and the clauses are labeled from left to right.

exactly one of the items o_u or $o_{\bar{u}}$ for $u \in Y$:

- The two items associated with the most significant digit whose label is in Y cannot be taken simultaneously in O_2 as it would violate the weight capacity A. In fact, exactly one of these items must be taken, as otherwise O_3 would select them both, making the achievement of the profit $\bar{K}$ only dependent on the items associated with the Z; consequently, the goal would be achieved.
- The two items associated with the second most significant digit whose label is in Y cannot be taken simultaneously, since we already know that one of the items associated with the most significant digit in Y is taken which would result in a violation of the weight capacity A. Hence, reasoning as before, O_2 will take exactly of the items associated with the second most significant digit in Y.
- The reasoning above propagates until the least significant digit labeled in Y. We conclude that the best O_2 will have exactly one of the items o_u or $o_{\bar{u}}$ for $u \in Y$.

Finally, O_3 will contain O_1 and all the items associated with Y not in O_2. This makes the rest of the items selection for O_3 completely equivalent to variable assignment in Z for $B_3 \cap CNF$ (precisely, the standard reduction from 3-SAT to Subset Sum). Therefore, TIK is a *Yes* instance.

Next, suppose that TIK is a *Yes* instance. Certainly, an optimal O_1 must have exactly one of the items o_u and $o_{\bar{u}}$ for $u \in X$, otherwise, O_2 could interdict some o_u and $o_{\bar{u}}$, making the goal $\bar{K}$ impossible to be achieved. As argued before, an optimal reaction O_2 to O_1 will select the items associated with X not in O_1.

Assign 1 to $u \in X$ such that $o_u \in O_1$, and 0 otherwise. For any valid assignment of the variables in Y, the correspondence in TIK is the following: if $u \in Y$ is 1, add $o_{\bar{u}}$ to O_2, otherwise add o_u. This forces O_3 to select for each $u \in Y$, o_u if u is 1 and $o_{\bar{u}}$ if u is 0; otherwise, the goal $\bar{K}$ is not attained. Since, by hypothesis, TIK is a *Yes* instance, for those O_1 and O_2, there is O_3 such that the profit $\bar{K}$ is exactly achieved which implies that there is an assignment of Z such that E is satisfied. □

Theorem 2.3.14. *MCN$_w$ is Σ_3^p-complete, even on trees.*

Proof. MCN$_w$ is clearly in Σ_3^p. Next, from an instance of TIK, we construct the following instance of MCN$_w$:

- Let $\Omega = A'$, $\Phi = A + 1$, $\Lambda = B$ and $K = \bar{K}$.
- For each item $o \in O$ create three nodes v_o^1, v_o^2 and v_o^3 with
  - $\hat{c}_{v_o^1} = \Omega + 1$, $h_{v_o^1} = \Phi + 1$, $c_{v_o^1} = p_o$ and $b_{v_o^1} = 0$; this node is only available for the protection set P;

  - $\hat{c}_{v_o^2} = \Omega + 1$, $h_{v_o^2} = \Phi + 1$, $c_{v_o^2} = \Lambda + 1$ and $b_{v_o^2} = p_o$; this node cannot be vaccinated, directly infected or protected;
  - $\hat{c}_{v_o^3} = a'_o$, $h_{v_o^3} = a_o$, $c_{v_o^3} = \Lambda + 1$ and $b_{v_o^3} = 0$; this node is only available for the vaccination set D and for the direct infection set I;
- Create a node r with $\hat{c}_r = \Phi + 1$, $h_r = 1$, $c_r = 1$ and $b_r = K$.
- For each item $o \in O$, add the edges (r, v_o^1), (v_o^1, v_o^2) and (v_o^2, v_o^3).

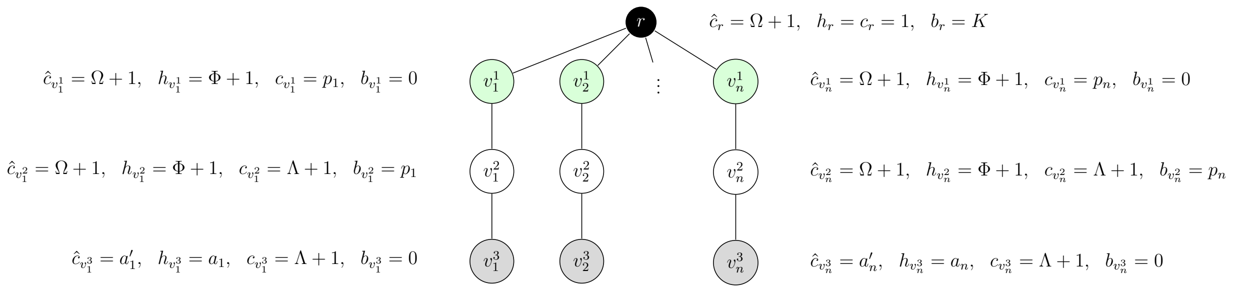

r
$\hat{c}_r = \Omega + 1, \quad h_r = c_r = 1, \quad b_r = K$
$\hat{c}_{v_1^1} = \Omega + 1, \quad h_{v_1^1} = \Phi + 1, \quad c_{v_1^1} = p_1, \quad b_{v_1^1} = 0$
v_1^1
v_2^1
$\vdots$
v_n^1
$\hat{c}_{v_n^1} = \Omega + 1, \quad h_{v_n^1} = \Phi + 1, \quad c_{v_n^1} = p_n, \quad b_{v_n^1} = 0$
$\hat{c}_{v_1^2} = \Omega + 1, \quad h_{v_1^2} = \Phi + 1, \quad c_{v_1^2} = \Lambda + 1, \quad b_{v_1^2} = p_1$
v_1^2
v_2^2
v_n^2
$\hat{c}_{v_n^2} = \Omega + 1, \quad h_{v_n^2} = \Phi + 1, \quad c_{v_n^2} = \Lambda + 1, \quad b_{v_n^2} = p_n$
$\hat{c}_{v_1^3} = a'_1, \quad h_{v_1^3} = a_1, \quad c_{v_1^3} = \Lambda + 1, \quad b_{v_1^3} = 0$
v_1^3
v_2^3
v_n^3
$\hat{c}_{v_n^3} = a'_n, \quad h_{v_n^3} = a_n, \quad c_{v_n^3} = \Lambda + 1, \quad b_{v_n^3} = 0$

Figure 2.9. Graph reduction from TIK to MCN$_w$ when $O = \{1,2\ldots,n\}$. The only nodes resulting in positive benefit are the ones in white. The nodes in gray can be vaccinated and directly attacked. The nodes in green can be protected. The node in black can be attacked (and protected).

See Figure 2.9 for an illustration of our reduction. Its key ingredients are the following: *(i)* independently of the vaccination strategy, an optimal attack will always include the node r, *(ii)* hence, the only way to collect a positive benefit p_o is by ensuring that node v_o^2 is saved, *(iii)* the latter is only possible if v_o^3 is vaccinated and v_o^1 is protected or if v_o^3 is not attacked and v_o^1 is protected. These observations allow to show that TIK is a *Yes* instance if and only if MCN$_w$ is a *Yes* instance. The remainder of the proof follows a similar reasoning to the previous proofs for the weighted games. □

2.3.4. Directed graphs

In this section, we consider directed graphs $G = (V,A)$ and restrict costs and benefits to be unitary. We use the subscript $_{dir}$ for these problem versions. Clearly, these problems inherit the complexity of their unitary undirected versions, as they are more general. In fact, we were able to go a level up in the polynomial hierarchy for some of its subgames in comparison with the unitary undirected cases. In this section, we first prove that the Attack$_{dir}$ is NP-complete, and then demonstrate that Vaccination-Attack$_{dir}$ is Σ_2^p-complete. Later, in Section 2.3.5, we present special properties of Protect$_{dir}$ that allow us to easily prove NP-completeness for directed acyclic graphs and polynomiality for arborescences.

It should be remarked that we do not address ATTACK-PROTECT$_{dir}$ and thus, it remains open whether it is Σ_2^p-complete. The difficulty on dealing with this subgame is related to the lack of Σ_2^p-hard problems involving unitary parameters or a division on the two players decision variables: in ATTACK-PROTECT$_{dir}$ all parameters are 1 and all nodes can be subject to infection or protection. On the other hand, as an example, non-trivial instances of KIP (presented in Section 2.3.3.2) should have weights not all 1, otherwise it becomes polynomially solvable as it can be reduced to its continuous version and, consequently, efficiently solved [Carvalho et al., 2018]. Another example, 2-CNF-ALTERNATING QUANTIFIED SATISFIABILITY, to be introduced in Section 2.3.4.2, and which is Σ_2^p-complete, demands each player to control distinct sets of variables. For VACCINATION-ATTACK$_{dir}$, we were able to bypass this challenge but an analogous trick does not seem easily adaptable for ATTACK-PROTECT$_{dir}$.

2.3.4.1. The ATTACK$_{dir}$ problem

First, we study the *Attack problem* on directed graphs, ATTACK$_{dir}$. We are given a directed graph G_a resulting from the deletion of the vaccinated nodes from the original graph, and an integer budget Φ. In this setting, there is no protection phase, *i.e.* $\Lambda = 0$. The decision version of the problem is:

ATTACK$_{dir}$:
INSTANCE: A directed graph $G_a = (V_a, A_a)$, a non-negative integer budget $\Phi \leq |V_a|$, and a non-negative integer K.
QUESTION: Is there a subset of nodes $I \subseteq V_a$, $|I| \leq \Phi$ such that the number of infected nodes in G_a is greater or equal to K?

We saw that in the undirected case, this problem is solvable in linear time, the best strategy being to infect the Φ largest connected components of G_a. But in the directed case, the infection is only allowed to propagate itself according to the direction of the arcs, which makes the problem of choosing the right set of nodes to attack NP-complete. We will use a reduction from the 3-*Satisfiability problem*, which is one of the Karp's 21 NP-complete problems [Karp, 1972].

3-SATISFIABILITY (3-SAT):
INSTANCE: Set U of variables, Boolean expression E over U in conjunctive normal form with exactly 3 literals in each clause $c \in C$.
QUESTION: Is there a 0-1 assignment for the variables in U that satisfies E?

Theorem 2.3.15. *ATTACK$_{dir}$ is NP-complete, even on directed acyclic graphs.*

PROOF. $\textsc{Attack}_{dir} \in$ NP as, given a set of attacked nodes I, checking whether the set of infected nodes is greater than K is easily done using a DFS.
To prove that $\textsc{Attack}_{dir}$ is NP-hard, we take an instance of 3-SAT. We build a directed acyclic graph G_a as follows:

- For each variable $u \in U$, we create two nodes v_u and $v_{\bar{u}}$, one for each possible 0-1 assignment of u. We call $V_U = \{v_u; u \in U\}$ and $V_{\bar{U}} = \{v_{\bar{u}}; u \in U\}$ the two sets of nodes of size $|U|$. For each variable u, we also create a directed path p_u of length $|C| + |U| - 1$, with an in-going arc from both v_u and $v_{\bar{u}}$ at the beginning of the path.
- For each clause $c \in C$, we create a node $v_c \in V_C$.
- From each node $v_u \in V_U$, we draw an arc (v_u, v_c) to every clause in which the positive literal u appears. Similarly, we draw an arc $(v_{\bar{u}}, v_c)$ from each $v_{\bar{u}} \in V_{\bar{U}}$ to every clause in which the negative literal $\neg u$ appears.

An example of this construction can be found in Figure 2.10. We set $\Phi = |U|$, $K = |U| \times (|U| + |C|) + |C|$ and argue that answering $\textsc{Attack}_{dir}$ on this instance is the same as answering 3-SAT.

Indeed, suppose that 3-SAT is a *Yes* instance, *i.e.* there is a 0-1 assignment to the variables in U such that every clause in E is true. Taking this assignment, by attacking v_u if u is set to be 1 and $v_{\bar{u}}$ otherwise, we attack exactly Φ nodes in G_a. Moreover, each path p_u is infected, and for each pair $(v_u, v_{\bar{u}})$, there is exactly one node infected due to the direction of the arcs. Finally, as E is true, each clause c is true, which translates into the fact that each v_c in the graph G_a is infected. Overall, there are exactly $|U| + |U| \times |p_u| + |C| = |U| \times (|U| + |C|) + |C|$ nodes infected in the graph.

Conversely, we prove that if $\textsc{Attack}_{dir}$ is a *Yes* instance, *i.e.*, there is a feasible attack I^* on G_a leading to at least $K = |U| \times (|U| + |C|) + |C|$ nodes infected, then E is satisfiable and the corresponding 0-1 assignment can be read in I. Let I^* be such an attack strategy. First, we remark that the largest possible set of infected nodes should contain all the nodes V_{p_u} of each path p_u: it is possible to infect them all as $\Phi = |U|$ and due to their size equal to $|C| + |U| - 1$, we can prove that not infecting all of them results in a sub-optimal solution. Indeed, suppose that for one u' we do not infect any of the nodes $V_{p_{u'}}$ of the path $p_{u'}$. Let α^* be the maximum number of nodes we can infect without infecting $p_{u'}$. As $p_{u'}$ is not infected, $v_{u'}$ and $v_{\bar{u}'}$ cannot be either. Thus, an easy upper bound α_{up} on α^* is obtained by saying that every node of the graph is infected, except for the ones in $\{v_{u'}, v_{\bar{u}'}\} \cup V_{p_{u'}}$. Then,

$$\begin{aligned}\alpha^* \leq \alpha_{up} &= (|U|-1) \times |p_u| + 2(|U|-1) + |C| \\ &= (|U|-1) \times (|U|+|C|-1) + 2|U| - 2 + |C| \\ &= |U|^2 + |U| \times |C| - 2|U| - |C| + 1 + 2|U| - 2 + |C| \\ &= |U| \times (|U|+|C|) - 1.\end{aligned}$$

As we assumed that the optimal attack I^* infected at least $K = |U| \times (|U|+|C|)+|C|$ nodes, which is strictly greater than α_{up}, we proved that no strategy not infecting all the paths can infect K nodes.

Thus, as there is exactly Φ different paths, we should attack exactly one element in each set of nodes $\{v_u, v_{\bar{u}}\} \cup V_{p_u}$: if we attacked more than one, then the remaining budget would not allow to attack all the paths. As attacking v_u or $v_{\bar{u}}$ leads to a strictly greater number of infected nodes than infecting a node in p_u, there is no harm in assuming that no node inside the p_u is in I^*. This implies that $I^* \subset V_U \cup V_{\bar{U}}$. At this point, there are at least $|p_u| \times |U| + |U| = |U| \times (|U|+|C|)$ nodes infected. Since we supposed that we had a *Yes* instance to ATTACK$_{dir}$, there must be $K = |U| \times (|U|+|C|) + |C|$ infected nodes, which implies that all nodes in V_C are infected. Thus, 3-SAT is a *Yes* instance and I^* is a 0-1 assignment of U that makes E true, concluding the proof. □

Remark 2.3.16. *Note that the proof of Theorem 2.3.15 holds if p_u is replaced by a complete graph with $|C|+|U|-1$ nodes (the length of the path). This observation will be useful for the reduction used in* VACCINATION-ATTACK$_{dir}$

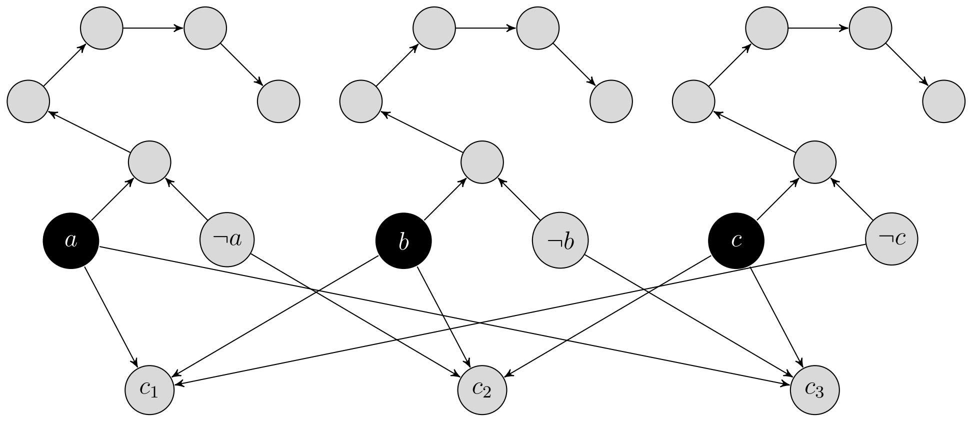


Figure 2.10. Example of construction of ATTACK$_{dir}$ from an instance of 3-SAT constituted of the boolean expression in CNF with 3 literals in each clause $E = (a \vee b \vee \neg c) \wedge (\neg a \vee b \vee c) \wedge (a \vee \neg b \vee c)$. We have $U = \{a,b,c\}$ and $|C| = 3$. Taking $I = \{v_a, v_b, v_c\}$ is optimal.

2.3.4.2. The Vaccination-Attack$_{dir}$ problem

Our demonstration of NP-completeness for Attack$_{dir}$ inspires our proof for the Σ_2^p-completeness of Vaccination-Attack$_{dir}$. The formulation of this decision problem is:

Vaccination-Attack$_{dir}$:
instance: A graph $G = (V,A)$, two non-negative integer budgets Ω and Φ such that $\Omega + \Phi \leq |V|$ and a non-negative integer K.
question: Is there a subset $D \subseteq V$, $|D| \leq \Omega$ such that $\forall I \subseteq V \backslash D$ with $|I| \leq \Phi$, the number of infected nodes $|V| - s(G, D, I, \emptyset) \leq K$?

We will use a reduction from the variant of the *2-Alternating Quantified Satisfiability problem* (B_2) by Johannes 2011 we introduced in the Chapter 1:

2-CNF-Alternating Quantified Satisfiability (B_2^{CNF}):
instance: Disjoint non-empty sets of variables X and Y, Boolean expression E over $U = X \cup Y$ in conjunctive normal form with exactly 3 literals in each clause.
question: Is there a 0-1 assignment for X so that there is no 0-1 assignment for Y such that E is satisfied?

Theorem 2.3.17. Vaccination-Attack$_{dir}$ *is* Σ_2^p*-complete.*

Proof. From the formulation in the form of $\exists D \ \forall I \ \ Q(D,I)$, we deduce that Vaccination-Attack$_{dir} \in \Sigma_2^p$.
To show that it is Σ_2^p-hard, we take an instance of B_2^{CNF}. We build G in a similar fashion to how G_a was built in the proof of the Theorem 2.3.15, the main difference being the use of cliques instead of paths. However, to differentiate the variables in X from the ones in Y, we slightly change the construction:

- For each variable $x \in X$, we create two nodes v_x and $v_{\bar{x}}$, one for each possible 0-1 assignment of x. We call V_X and $V_{\bar{X}}$ the sets of v_x and $v_{\bar{x}}$. We also create two cliques k_x and $k_{\bar{x}}$ of $|C| + |Y| - 1$ nodes V_{k_x} and $V_{k_{\bar{x}}}$.
- For each variable $y \in Y$, we create two nodes v_y and $v_{\bar{y}}$, one for each possible 0-1 assignment of y. Let V_Y and $V_{\bar{Y}}$ be these two sets of nodes, and $V_U = V_X \cup V_Y$, $V_{\bar{U}} = V_{\bar{X}} \cup V_{\bar{Y}}$. We also create a clique k_y of size $|C| + |Y| - 1$.
- For each clause $c \in C$, we create a node $v_c \in V_C$.
- From each node $v_u \in V_U$, we draw an arc (v_u, v_c) to every clause in which the positive literal u appears. Similarly, we draw an arc $(v_{\bar{u}}, v_c)$ from each $v_{\bar{u}} \in V_{\bar{U}}$ to every clause in which the negative literal $\neg u$ appears.

— From every v_x, we draw an arc to one node in k_x, and do the same thing with $v_{\bar{x}}$ and $k_{\bar{x}}$. We also draw an undirected edge between each v_x and $v_{\bar{x}}$.

— Finally, from each v_y and each $v_{\bar{y}}$, we draw an arc to one node in k_y.

An example of this construction can be found in Figure 2.11. We set $\Omega = |X|$, $\Phi = |X| + |Y|$, $K = (|X| + |Y|) \times (|Y| + |C|) + |C| - 1$ and argue that answering VACCINATION-ATTACK$_{dir}$ on this instance is the same as answering B_2^{CNF}.

Indeed, if we are a given a solution to a *Yes* instance of B_2^{CNF}, then by vaccinating the nodes corresponding to the opposite of the 0-1 assignment of X, we oblige the attacker to infect the nodes corresponding to the truth values for X. From there, by following the same reasoning as before, it is easy to see that the *Yes* instance of B_2^{CNF} leads to a *Yes* instance of VACCINATION-ATTACK$_{dir}$, *i.e.* the attacker cannot infect more than K nodes.

Conversely, we show that a set D^* corresponding to a solution of a *Yes* instance of VACCINATION-ATTACK$_{dir}$ is a solution to a *Yes* instance of B_2^{CNF}. The first thing to notice is that given that the vaccination budget is $\Omega = |X|$, that the size of the cliques k_x and $k_{\bar{x}}$ is equal to $|C| + |Y| - 1$ and that each clique can be disconnected from the graph by spending only one unit of vaccination budget, we necessarily have that the best vaccination strategy $D^* \subset \bigcup_{x \in X} \{v_x, v_{\bar{x}}\}$. Next, we show that the defender would be worse off is she decides to vaccinate both $v_{x'}$ and $v_{\bar{x}'}$ for some $x' \in X$ instead of vaccinating exactly one of each member of $\{v_x, v_{\bar{x}}\}$. In the best case scenario, in addition to the nodes already vaccinated, deciding to vaccinate the two members of a pair will allow her to protect $|C| - 1$ nodes in V_C *(it is not possible to remove all the arcs between the $V_U \cup V_{\bar{U}}$ and the V_C as we suppose that $Y \neq \emptyset$, thus at least one clause contains a variable from Y)*. But by doing so, as $\Omega = |X|$, the defender will also not protect at all a group of nodes $\{v_{x''}, v_{\bar{x}''}\} \cup V_{k_{x''}} \cup V_{k_{\bar{x}''}}$. Thus, the attacker can then spend only one unit of her own budget to attack all of this group, a quantity of infected nodes that otherwise would have been obtained by spending two units of his budget Φ. Thus, defending the two members of $\{v_{x'}, v_{\bar{x}'}\}$ spared one unit of budget for the attacker, which she can then use to attack one of the disconnected cliques of size $|C| + |Y| - 1 > |C| - 1$. Thus, making such a move for the defender is strictly worse than not doing it and D^* contains exactly one node from each $\{v_x, v_{\bar{x}}\}$.
After this stage, it is easy to see that the best move for the attacker is to attack all of the $D^* \backslash (V_x \cup V_{\bar{x}})$, and for the variables in Y, the situation reduces to the one we already discussed with ATTACK$_{dir}$ *(note that it is always more interesting for the attacker to spend her budget on attacking the v_y and $v_{\bar{y}}$ than the disconnected cliques as it will always infect more nodes)*. Hence, in the end, if the attacker did not manage to infect strictly more than $(|Y| + |X|) \times (|Y| + |C|) + |C| - 1$ nodes, it means that at least one clause is false, which concludes the proof. □

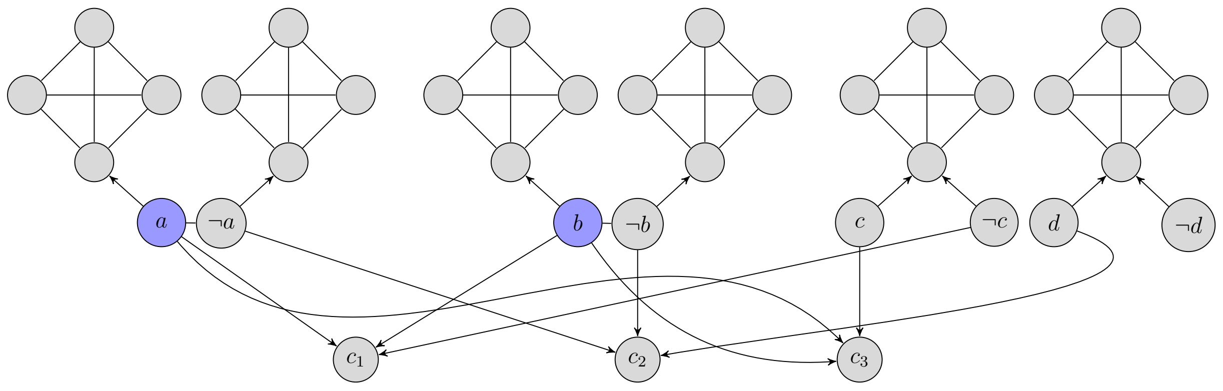


Figure 2.11. Example of construction of VACCINATION-ATTACK$_{dir}$ from an instance of B_2^{CNF} constituted of the boolean expression in CNF with 3 literals in each clause $E = (a \lor b \lor \neg c) \land (\neg a \lor \neg b \lor d) \land (a \lor c \lor b)$. Here, $X = \{a, b\}$ and $Y = \{c,d\}$. Taking $D = \{v_a, v_b\}$, i.e obliging both a and b to be *False* makes it impossible to satisfy E.

Corollary 2.3.18. MCN$_{dir}$ *is* Σ_2^p*-hard.*

2.3.5. PROTECTION$_{dir}$: tractability limits

In this section, we will concentrate on optimal protection strategies given I (directly infected nodes). Without loss of generality, in what follows, we are restricting our attention to the induced graph obtained by only considering non-saved nodes when there is no protection.

The motivation to provide a closer look to the protection problem in the directed case is based on the fact that its NP-completeness was established for split graphs in the unitary case and for trees in the weighted one. Such results do not clarify the problem complexity for directed acyclic graphs (DAGs), polytrees and arborescences. Frequently, NP-complete problems on graphs became polynomially solvable on such graph classes. In this section, we start by focusing on general results for directed acyclic graphs and then on arborescences.

2.3.5.1. Directed acyclic graphs

We will show that an optimal protection strategy can be restricted to *candidate* nodes for any directed acyclic graph.

Definition 2.3.19. *In a directed graph $G = (V,A)$, a node $v \in V \setminus I$ that can be reached from a node of I by a directed path and whose isolated protection results in a maximal set of strongly connected saved nodes, is called candidate. Denote by $\mathcal{C}$ the set of candidate nodes.*

In other words, a candidate node v has no predecessor that implies saving v. See Figure 2.13a for an illustration: $\mathcal{C} = \{1,2,3,9\}$; *e.g.*, node 5 is not a candidate, since its protection saves nodes $\{6,7,8\}$, but this is also guaranteed by saving node 2 instead, resulting in the maximal set of saved nodes $\{2,3,4,5,6,7,8\}$.

Lemma 2.3.20. *Let $G = (V,A)$ be a directed acyclic graph. Given I and Λ, there is an optimal protection strategy $P \subseteq \mathcal{C}$.*

Proof. Let $P \subseteq V \setminus I$ be an optimal protection strategy such that exists $v \in P \setminus \mathcal{C}$. Then, by definition of candidate, there is a node $u \in \mathcal{C}$ whose isolated protection implies saving v, as well as, all the nodes that v alone was saving. Hence, a feasible protection strategy can be obtained by removing v from P and adding u to P: note that either the used budget is maintained, if $u \notin P$, or decreased, if $u \in P$. Let this strategy be denoted by $\tilde{P} = (P - \{v\}) \cup \{u\}$.

By contradiction, suppose that $\tilde{P}$ is not optimal: there is some node r that was saved in P but not in $\tilde{P}$. In fact, we can conclude that under P, r was saved due to v being saved and possibly due to some other nodes in $P \setminus \{v\} \subseteq \tilde{P}$. However, under $\tilde{P}$, v is also saved, as well, as the nodes in $P \setminus \{v\}$. Consequently, r is saved in $\tilde{P}$, resulting in a contradiction. $\square$

Furthermore, we can compute the *value* of candidate nodes.

Definition 2.3.21. *For each $v \in \mathcal{C}$, the value of v is denoted by p_v and it corresponds to the number of saved nodes if v is the only protected node.*

In the example of Figure 2.13a, $p_1 = 1$, $p_2 = 6$, $p_3 = 1$ and $p_9 = 1$. However, note that this analysis does not make the problem trivial: in Figure 2.13a, if $\Lambda = 2$, the optimal protection cannot be computed in a greedy way, *i.e.*, protecting nodes 1 and 2 is not optimal; the only optimal solution is to protect nodes 1 and 3.

Remark 2.3.22. *Definitions 2.3.19 and 2.3.21, and Lemma 2.3.20 do not need the condition that G is a directed acyclic graph. Hence, they extend to any directed graph and thus, undirected graphs. However, since we know that the protection problem is NP-complete for general directed graphs, for sake of simplicity, we decided to write the full section assuming DAGs.*

Theorem 2.3.23. Protect$_{dir}$ *is NP-complete, even for directed acyclic graphs.*

Proof. The statement of Protect$_{dir}$ is exactly the one of Protect in Section 2.3.2, except that the graph is directed. For sake of simplicity, we drop the subscript a from G_a.

The problem is clearly in NP as given the protection P, the number of infected can be determined in polynomial time through a DFS.
Next, we reduce CNP$_{split}$ to PROTECT$_{dir}$, showing its NP-hardness. Given an instance of CNP$_{split}$, we build the following graph $G = (V,A)$:

— For each $v \in \bar{V}_1$, we create the set of nodes $T_v = \{t_v^1, t_v^2\}$ in G, and the arc (t_v^1,t_v^2).
— For each $v \in \bar{V}_2$, we replicate it in G, and for each edge $(r,v) \in \bar{E}$ with $v \in \bar{V}_2$, the arc (t_r^1,v) is added in G.
— Finally, we add the only attacked node u to G and connect it with each t_v^1 for $v \in \bar{V}_1$, through the arc (u,t_v^1).

To complete the reduction it remains to set $\Lambda = B$ and $K = \lfloor 2 + \sqrt{8\bar{K} + 1} \rfloor$ (obtained by solving $\bar{K} = \binom{\frac{K-1}{2}}{2}$). See Figure 2.12 for an illustration of the reduction.

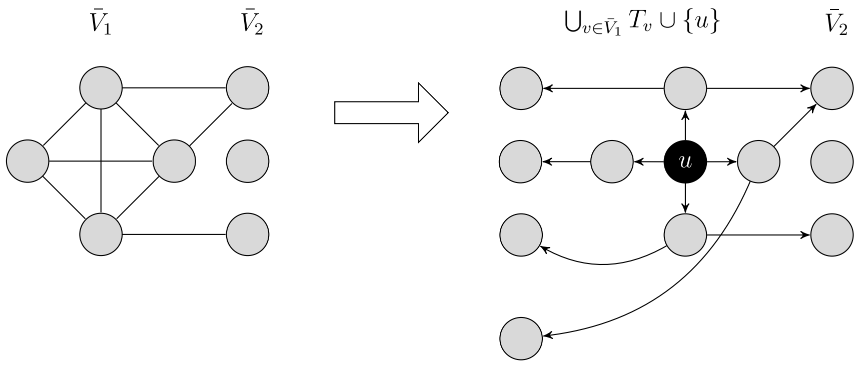


Figure 2.12. Graph reduction from CNP$_{split}$ to PROTECT$_{dir}$

First, note that $\mathcal{C}$ of G is $\{t_v^1 : v \in \bar{V}_1\} \cup \bar{V}_2$, where the nodes in the first set have value at least 2, and the ones in the second have value 1. Hence, it is clear that the best protection strategy will prioritize the nodes t_v^1. In fact, we can argue than only those nodes can be in an optimal protection strategy. If $\Lambda = B \geq \bar{V}_1$, then the instance of CNP$_{split}$ is trivial. Therefore, we can assume $\Lambda = B < \bar{V}_1$ and thus, it holds $P^* \subset \{t_v^1 : v \in \bar{V}_1\}$. Consequently, choosing the optimal P^* means to minimize the nodes in T_v, for $v \in \bar{V}_1$, and in $\bar{V}_2$ that are connected to u. By construction, those nodes connected with u correspond to a connected component $\bar{C}_1$ in $\bar{G}$. Thus, P^* minimizes the size of $\bigcup_{v\in\bar{C}_1} \{t_v^2\} \cup \{u\} \cup \bar{C}_1$. The remaining of the proof follows an analogous reasoning to the proof of Theorem 2.3.3. □

2.3.5.2. Arborescence

In this section we restrict the protection problem to the case where the graph induced by $V \setminus I$ is an arborescence.

Definition 2.3.24. *A DAG $G = (V,A)$ is an arborescence if its underlying undirected graph is a tree (forest) and there is a single node (root) that has a unique directed path from it to all other nodes.*

In arborescence, it is direct the determination of $\mathcal{C}$. Since all nodes in $V \setminus I$ have in-degree 1, either they are protected by their predecessor, and thus are not a candidate, or they are direct successors of nodes in I. Therefore, $\mathcal{C}$ is the set of all successors of nodes in I. For an illustration see Figure 2.13c. We can prove that in this case a greedy approach leads to optimality.

Lemma 2.3.25. *Given $G = (V,A)$, I and Λ, if the graph induced by $V \setminus I$ is an arborescence, then an optimal protection can be determined in polynomial time, specifically, $O(|V| \log(|V|))$. Moreover, if the induced graph is a set of arborescences, the result also holds.*

Proof. We start by showing that a greedy procedure runs in time $O(|V| \log(|V|))$. As previously observed, for arborescences, the set of candidate nodes is easy to compute: it is the set of all successors of I.

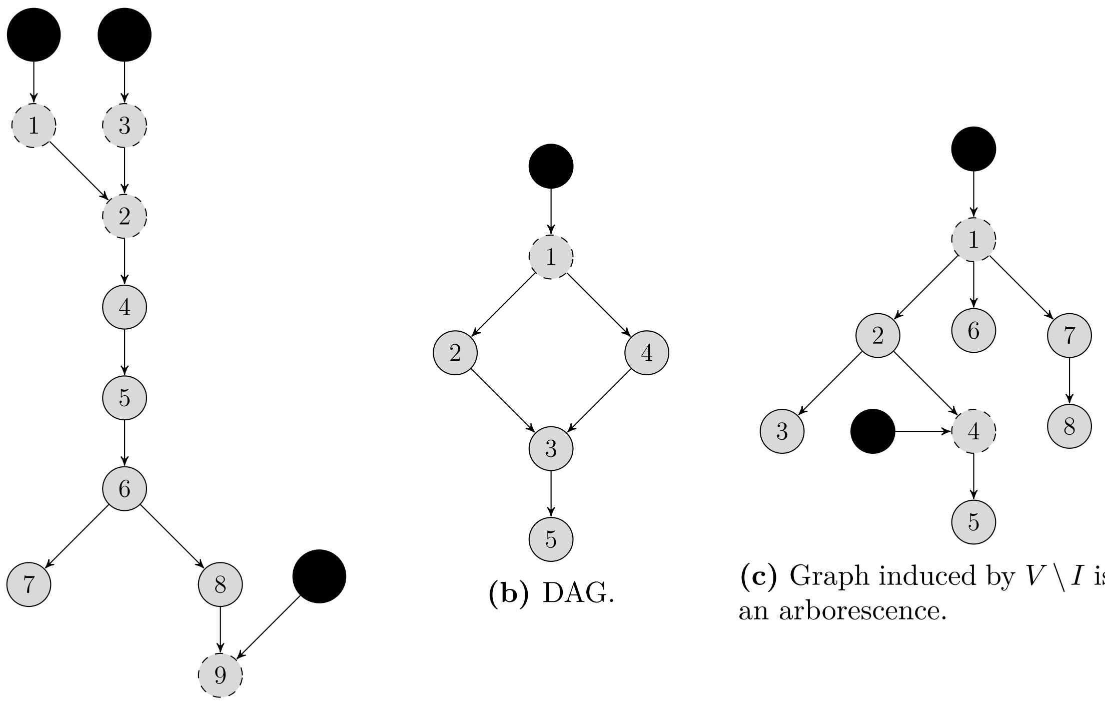
Next, the calculation of p_v for each $v \in \mathcal{C}$ can be performed through a depth-first-search that records the saved nodes by candidates. This requires $O(|V|)$ since the graph is an arborescence.

Finally, the Λ candidate nodes of largest values are protected. This requires to order the nodes accordingly with $\{p_v\}_{v \in \mathcal{C}}$. Thus, the greedy method runs in $O(|V| \log(|V|))$.

Next, we show that the described method provides an optimal protection. Let P be the obtained protection through the greedy method. The key idea to prove the optimality of P is essentialy due to the fact that in an arborescence, $\mathcal{C}$ is simply the set of all successors of I, otherwise, if we have a node of in-degree at least 2, we do not have an arborescence. Thus, the protection strategy P cannot imply the protection of some candidate not in P. This shows the optimality of P. $\square$

Note that in trees (undirected graphs), it does not hold that $\mathcal{C}$ is the set of successors of the nodes in I. Hence, Lemma 2.3.25 does not extend to the undirected case.

Remark 2.3.26. *Note that in Lemmata 2.3.20 and 2.3.25, we did not used the fact that $b_v = 1$. Thus, these lemmas also holds when nodes benefits are not unitary.*



(a) Polytree. **(b)** DAG. **(c)** Graph induced by $V \setminus I$ is an arborescence.

Figure 2.13. The set I is represented by black nodes and candidate nodes are dashed.

2.4. Conclusion

In this chapter, we shed light on the practical challenges encountered when exactly solving MCN and its subgames, and explained them by providing new results on their computational complexities. We showed that MCN is at least NP-hard in the undirected unitary case, at least Σ_2^p-hard in the directed unitary one and Σ_3^p-complete in the weighted case. These results motivate the design of efficient heuristics to solve this multilevel combinatorial problem. In particular, we focus in this thesis on how one can *learn* to find such heuristics thanks to Graph Neural Networks. In order to rigorously frame this problem in a learning context, we will need to use a Multi-Agent Reinforcement Learning setting. In the next chapter, we establish the necessary background to properly introduce the learning frameworks we will discuss in Chapter 4.

Chapter 3

Learning heuristics for Combinatorial Optimization problems

In previous chapters, we introduced the Multilevel Critical Node Problem (MCN) and showed that this two-player sequential game played on a graph is provably hard to solve, assuming that P $\neq$ NP. We also showed that this complexity expresses itself in practice as it is challenging to find a solution to an instance of the problem with the currently existent exact solver. Thus, the MCN belongs to the vast list of NP-hard combinatorial optimization problems over graphs that quickly become intractable to solve with exact methods. To deal with those practical issues that arise in Operations Research, practitioners often have to come to terms with finding only sub-optimal solutions when tackling real-world instances of combinatorial problems. To do so, heuristics are widely used in practice as they can be engineered to be highly scalable, but at the expense of having no theoretical guaranty on the quality of the solutions produced. Hence, the design of powerful heuristics that empirically approach optimality has attracted the attention of many Computer Scientists over the years [Gonzalez, 2007].

With recent advances in Deep Learning [Goodfellow et al., 2016] and Graph Neural Networks (GNN) [Wu et al., 2020], the idea of leveraging the recurrent structures appearing in the combinatorial objects belonging to a *distribution of instances* of a given problem to learn efficient heuristics with a Reinforcement Learning (RL) framework has received an increased interest [Bengio et al., 2018; Mazyavkina et al., 2020]. In this thesis, we decided to take this direction and designed a Reinforcement Learning framework in order to learn to solve the MCN with a Graph Neural Network based agent. In this chapter, we will present the necessary background to introduce the setting we will discuss in Chapter 4. We will begin by introducing the notion of *Graph Neural Network* in Section 3.1. Then, in Section 3.2, we will discuss how one can learn to solve sequential games by presenting the *Reinforcement Learning* and *Multi-Agent Reinforcement Learning* frameworks. Finally, we will bring together GNN and RL in Section 3.3 by reviewing some previous work on *Deep Learning for Combinatorial Optimization* on which the novel method we designed to learn to solve the MCN is built up.

3.1. Introduction to Graph Neural Networks

We aim to learn vector representations of graphs that could allow an agent to systematically find the optimal decisions to take in a given instance of a combinatorial optimization problem. The main challenge is thus to find a way to include information about the graph structures into a machine learning model. However, there is no straightforward way to encode into a single feature vector all the high-dimensional, non-Euclidean information that is enclosed inside a given graph structure. To tackle this issue, a method extending deep learning approaches for graph data have recently emerged under the name of *Graph Neural Networks*.

The main advantage of Deep Learning is to learn useful representations for the downstream machine learning task directly from raw data, while traditional Machine Learning techniques require the additional work of hand-crafting features to represent data with vectors. Thus, instead of using engineered features, we seek to learn mappings embedding nodes, or entire graphs structures, into points of a low dimensional vector space $\mathbb{R}^d$ so that the geometric relationship between those points reflects the structure of the graph. However, while there is a principled way to create Deep Learning models dealing with images or sequences thanks to *Convolutional Neural Networks* (CNN) and *Recurrent Neural Networks* (RNN), their flexible topology and intrinsic absence of notion of *order* in their sub-parts make graphs a challenging data structure to tackle. For example, the naive approach of seeing a graph as a binary image through its adjacency matrix is not permutation invariant, thus, directly applying a CNN to the binary image or flattening it and feeding it to a multilayer perceptron will result in embeddings that will depend on the arbitrary ordering of nodes we used to create the adjacency matrix, which is not an intrinsic property of the graph structure. In order to circumvent this issue, a parameterized permutation invariant operation is at the root of many Graph Neural Networks: the *Neural Message Passing*.

3.1.1. Neural Message Passing

Here, we focus on the problem of embedding *nodes* of a graph. In the next section, we present how we can go from node representations to graph level ones. More particularly, we aim to embed nodes in a way that reflects their *role* inside a given graph, so that an agent taking decisions at the node level can easily identify the ones playing the same role for the task at hand and classify them by their relevance to him/her. In the case of the MCN, if the agent is the protector, we seek to identify the nodes that are the most interesting to protect. In general, we want to find a way to represent nodes that both take into account their *structural role* inside the graph, *i.e.*, the position they occupy inside the graph's topology, and also contain information about the possible *attributes* they have with respect to the attributes of the other nodes in the graph. For example, in the case of the MCN, an attribute attached to a node could be whether this node has been attacked or not. In order to formalize the setting, we introduce $\mathbf{A} \in \{0,1\}^{n\times n}$ the adjacency matrix of the graph $G = (V,A)$ with n nodes, and $\mathbf{X} \in \mathbb{R}^{n\times p}$ the matrix containing the p attributes of each node. From this, our purpose is thus to create a new matrix $\mathbf{H} \in \mathbb{R}^{n\times d}$ that embeds in a relevant way the n nodes in $\mathbb{R}^d$.

A promising way to tackle such task is through Graph Neural Networks (GNN). In order to build representations that encode the two types of information we are interested in, *i.e.* the *structural* and *feature-based* information of the nodes, GNNs use the *Neural Message Passing* algorithm. The intuition behind this algorithm is to start from the node's features ${\mathbf{h_v}}^{(0)} = \mathbf{x_v}$ for $v \in V$ (and $\mathbf{x_v}$ the associated row of $\mathbf{X}$), and iteratively update them by aggregating the information of each node's neighbors $\mathcal{N}(v) = \{u \in V : (u,v) \in A\}$. Thus, with one iteration, the embedding ${\mathbf{h_v}}^{(1)}$ of each node $v \in V$ also contains information about its direct neighbors. From there, if we update once again each node's embedding by aggregating the embeddings of its local neighborhood, the embeddings $\{{\mathbf{h_v}}^{(2)}\}_{v\in V}$ now contain information about their 2-hop neighbors. In fact, after k iterations of the aggregation-update procedure, the embeddings $\{{\mathbf{h_v}}^{(k)}\}_{v\in V}$ contain information about their k-hop neighbors. The overall procedure can be summarized as in [Hamilton, 2020] with the following iterative equation:

$$ {\mathbf{h_v}}^{(k+1)} = \text{UPDATE}^{(k)}\left({\mathbf{h_v}}^{(k)}, \text{AGGREGATE}^{(k)}\left(\{{\mathbf{h_u}}^{(k)},\ \forall u \in \mathcal{N}(v)\}\right)\right). \tag{3.1.1} $$

Different motivations have lead to formulations falling under this general equation: Bruna et al. 2014 derived one using graph signal processing tools to generalize Euclidean convolutions to the non-Euclidean graph domain, Dai et al. 2016 established a connection between the task of embedding latent variables in structured data and *probabilistic graphical model* inference procedures such as mean field and belief propagation, while Hamilton et al. 2017 took inspiration from the Weisfeiler-Lehman graph isomorphism test to propose their GraphSAGE algorithm.

We remark in Equation 3.1.1 that, once again, we want to find a way to represented a *set*, *i.e.*, an *unordered* structure, into a single vector through the function AGGREGATE. Thus, the aggregation function must be permutation invariant as there is no natural ordering of the node's neighbors. To do that, Zaheer et al. 2017 state the following theorem:

Theorem 3.1.1. *[Zaheer et al., 2017] Assume the elements are from a compact set in $\mathbb{R}^p$, i.e., possibly uncountable, and the set size is fixed to n. Then any continuous function operating on a set X, i.e., $f : \mathbb{R}^{p\times n} \to \mathbb{R}$ which is permutation invariant to the elements in X can be approximated arbitrarily close in the form of $\rho(\sum_{x\in X} \phi(x))$, for suitable transformations ϕ and ρ.*

In particular, if ϕ and ρ are *universal approximators* such as multilayer perceptrons, the theorem holds [Zaheer et al., 2017]. Thus, using neural networks in the AGGREGATE function allows us to *learn* a suitable permutation invariant function for our use case. In particular, in the next chapter, we will use a *Graph Attention Network* (GAT) [Velicković et al., 2018] which aggregates node embeddings through a conical combination of linear projections defined with the following equations:

$$\mathbf{h_v}^{(k+1)} = \mu_{v,v}^{(k)}\boldsymbol{\Theta}^{(k)}\mathbf{h_v}^{(k)} + \sum_{u\in\mathcal{N}(v)} \mu_{v,u}^{(k)}\boldsymbol{\Theta}^{(k)}\mathbf{h_u}^{(k)} \tag{3.1.2}$$

with μ defined by:

$$\mu_{v,u} = \frac{\exp(\text{LeakyReLU}(\mathbf{a}^\top[\boldsymbol{\Theta}\mathbf{h_v}||\boldsymbol{\Theta}\mathbf{h_u}]))}{\sum_{k\in\mathcal{N}(v)\cup\{v\}} \exp(\text{LeakyReLU}(\mathbf{a}^\top[\boldsymbol{\Theta}\mathbf{h_v}||\boldsymbol{\Theta}\mathbf{h_k}]))}, \tag{3.1.3}$$

where $\mathbf{a} \in \mathbb{R}^{2\times d_{k+1}}$ and $\boldsymbol{\Theta} \in \mathbb{R}^{d_{k+1}\times d_k}$ are the trainable parameters. Here, d_k is the original embedding dimension of $\mathbf{h_v}^{(k)}$, d_{k+1} is the dimension of $\mathbf{h_v}^{(k+1)}$.

Through Equation 3.1.2, we notice that the UPDATE function used in GAT is another conical combination between the aggregated vector and a projection of the previous version of the node embedding. While such basic types of update methods are usual in GNNs, they can lead to *over-smoothing* problems when many iterations are performed, *i.e.*, there is a risk to make all the node representations similar to one another [Hamilton, 2020]. More complex methods exist to circumvent this issue; see the chapter 4.3 of [Hamilton, 2020] for a comprehensive survey on the matter.

3.1.2. Graph level representations

In the previous section, we introduced the Neural Message Passing method, aimed at creating node embeddings. However, in certain tasks, decision makers directly act on graphs or sub-graphs and not at the node level. Thus, there is also a need to create graph-level embeddings. There are two main ways to do that from the node embeddings. The first supposes that all the information about the graph's topology is already contained in the node embeddings we derived with a GNN. This implies that it suffices to see the graph as a *set* of node embeddings, meaning that any permutation invariant function applied on this set, such as the sum, the max, or a more evolved one using neural networks and Theorem

3.1.1, will lead to a fixed sized representation of the graphs. This is the method we decided to use in Chapter 4.

A more complex approach is to gradually simplify the graph structure by *coarsening* it iteratively, updating the node embeddings in the simplified structure, until the graph is summarized in one node. An example of such method is described in [Lee et al., 2019].

3.2. Introduction to Reinforcement Learning and Multi-Agent Reinforcement Learning

With Graph Neural Networks, we introduced a way to learn expressive node and graph representations. However, in order for them to be useful for the downstream task, we still need to define what would constitute a good learning signal to gradually refine the parameters of a GNN through gradient descent. In this thesis, we focus on a combinatorial optimization problem on a graph, thus, our aim is to take *optimal* decisions in any given instance. A straightforward way to learn to embed the combinatorial structures so that a decision maker can easily identify the optimal decisions would be to use the predominant paradigm in Machine Learning: supervised learning, or, in simpler words, learning by example. However, to do that, we would need a large dataset of examples, *i.e.*, pairs constituted of an instance and its corresponding optimal decisions. But constituting such a dataset is challenging for NP-complete combinatorial problems as it would necessitate to exactly solve a large quantity of instances, which cannot be done efficiently by definition, unless P=NP. In this context, the go-to framework to learn the representations we seek is *Reinforcement Learning* (RL).

In essence, RL consists in learning how to map situations to actions in order to maximize a numerical reward signal [Sutton and Barto, 1998]. Contrary to supervised learning where targets are available, RL focuses on agents that learn by interacting with the environment: they try different possible actions in order to discover the ones yielding the most rewards. A way to formally model this situation is by using *Markov Decision Processes* (MDPs), *e.g.*, [Puterman, 1994].

3.2.1. Markov Decision Process

The setting we are considering is constituted of a *learning agent* acting at each time step t in an *environment*. At time t, the agent is in a *state* $s_t \in \mathcal{S}$ of the environment and takes an *action* $a_t \in \mathcal{A}(s_t)$. By performing a_t in s_t, the agent changes the state of the environment which evolves to s_{t+1}. This transition leads to a *reward* r_t for the agent. At each time t, the purpose of the agent is to take an action that maximizes its long-term reward, which we formally define through the *discounted return* $G_t = \sum_{k \geq 0} \gamma^k r_{t+k}$ with $\gamma \in [0,1]$ the *discount rate*. A *Markov Decision Process* is then defined as the tuple $\langle \mathcal{S}, \mathcal{A}, P, R, \gamma \rangle$, with $\mathcal{S}$ the set of states, $\mathcal{A}$ the action space, P the transition function which defines the probabilities of the next state s_{t+1} given the current pair of state-action (s_t, a_t), the reward function $R : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$ and γ the discount rate.

In order to act, the agent follows a *policy* π, *i.e.*, from each perceived state s_t, the agent attributes a probability $\pi(a_t|s_t)$ to each of the possible actions $a_t \in \mathcal{A}(s_t)$. Under π, we can attribute a *value* $V_\pi(s)$ to each state s: the expected return starting from s and following π thereafter, *i.e.*, $V_\pi(s) = \mathbb{E}_\pi \left[\sum_{k \geq 0} \gamma^k r_{t+k} | s_t = s \right]$. Thus, the goal of an RL algorithm is to find an *optimal policy* π^* maximizing the value $V_\pi(s)$ for all initial state s. From there, the *optimal state-value function* V^* is defined as $V^*(s) = \max_\pi V_\pi(s)$. Similarly, we can attribute a value $Q_\pi(s,a)$ to the act of taking action a in state s under policy π. We call Q_π the *action-value function*, defined with $Q_\pi(s,a) = \mathbb{E}_\pi \left[\sum_{k \geq 0} \gamma^k r_{t+k} | s_t = s, a_t = a \right]$, and $Q^*(s,a)$ the *optimal action-value function* defined as $Q^*(s,a) = \max_\pi Q_\pi(s,a)$. Thus, we have the relation $V^*(s) = \max_{a \in \mathcal{A}(s)} Q^*(s,a)$. From these definitions emerge recursive links for the expected rewards called the *Bellman equations*. For example, the *Bellman optimality equation* for V^* is:

$$V^*(s) = \max_{a \in \mathcal{A}(s)} R(s,a) + \gamma \sum_{s'} P(s'|s,a) V^*(s') \quad \forall s \in \mathcal{S}. \tag{3.2.1}$$

In finite MDPs, with finite, small-sized sets $\mathcal{S}$ and $\mathcal{A}$, it is possible to use dynamic programming to find V^* from Equation 3.2.1. For example, in the *Value Iteration* method, the intuition is to see the Bellman optimality equation as an update rule until the estimates $V(s)$ of the values of the optimal value function converge for each possible state $s \in \mathcal{S}$. From there, π^* is obtained by simply following the greedy policy with respect to V^*.

Another way to solve the *control* task, *i.e.*, approximate an optimal policy, is to use *Monte-Carlo* samples. With this method, the values estimates are updated by comparing them to the return obtained by following the current version of the greedy policy during an *episode* $s_0, a_0, r_0, s_1, a_1, r_1 ..., s_T, a_T, r_T$, with T the last time step of the procedure.

A quicker way to update our estimates is to use a *Temporal Difference* method which combines both the dynamic programming and Monte Carlo approaches. Instead of waiting that a whole episode is finished to obtain an estimate of the target G_t as with Monte Carlo methods, Temporal Difference leverages the Bellman equation to produce the target by simply querying an estimate of the value of the *next state*. In Q-learning [Watkins, 1989], this results in the following update rule, with α a step-size $\in$ (0,1]:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1},a) - Q(s_t,a_t) \right] \tag{3.2.2}$$

However, these guaranteed to converge methods for finite MDPs do not directly generalize to situations with large, possibly infinite state spaces as it is not practicable anymore to visit each possible state. To tackle these situations, we need to learn to generalize the experience we previously had to new situations, *i.e.*, learn to estimate the value of an unseen state. To do that, we rely on function approximations for Q^* or V^* based on deep neural networks. Although the theoretical properties of such methods are not as strong nor well understood as the ones based on lookup tables [Sutton and Barto, 1998], using the same ideas to build update rules for the function approximation leads to great result in practice. For example, with *Deep Q-Networks* (DQN), Mnih et al. 2015 achieved great performances on 49 different Atari 2600 video games simply by interacting with a game emulator and using a neural network based agent trained to approximate the optimal Q function with the following update rule:

$$\mathbf{w_{t+1}} = \mathbf{w_t} + \alpha \left[r_{t+1} + \gamma \max_a \hat{Q}(s_{t+1},a, \mathbf{w_t}) - \hat{Q}(s_t, a_t, \mathbf{w_t}) \right] \nabla_{\mathbf{w_t}} \hat{Q}(s_t, a_t, \mathbf{w_t}) \tag{3.2.3}$$

where $\hat{Q}$ denotes the neural network and $\mathbf{w_t}$ is the vector of the network's weights at time t.

3.2.2. Alternating Markov Games

We have seen that with *Deep Reinforcement Learning*, we can design methods to make an agent learn good strategies for sequential optimisation problems with large state spaces. However, the formalism we used so far is not suited to the case where several agents coexist in the same environment. Whereas single agent RL is usually described with Markov Decision Processes, the framework needs to be extended to account for multiple agents. This has been done in the seminal work of Shapley 1953 by introducing *Markov Games* (also named *Stochastic Games*), which are at the base of the *Multi-Agent Reinforcement Learning* (MARL) setting [Littman, 1994; Shoham et al., 2007]. In the case of the MCN, we want to model two-player games in which moves are not played simultaneously but alternately. The natural setting for such situation was introduced by Littman in [Littman, 1994, 1996; Littman and Szepesvari, 1996] under the name of *Alternating Markov Games.*

An Alternating Markov Game involves two players: a maximizer and a minimizer. It is defined by the tuple $\langle \mathcal{S}_1, \mathcal{S}_2, \mathcal{A}_1, \mathcal{A}_2, P, R \rangle$ with $\mathcal{S}_i$ and $\mathcal{A}_i$ the set of states and actions, respectively, for player i, P the transition function mapping state-actions pairs to probabilities of next states and R a reward function. For $s \in \mathcal{S} = \mathcal{S}_1 \cup \mathcal{S}_2$, we re-define $V^*(s)$ as the expected reward of the concerned agent for following the optimal minimax policy against an optimal opponent starting from state s. In a similar fashion, $Q^*(s,a)$ is here the expected reward for the player taking action a in state s and both agents behaving optimally thereafter. Finally, with the introduction of the discount factor γ, we can write the generalized Bellman equations for Alternating Markov Games [Littman, 1996]:

$$V^*(s) = \begin{cases} \max_{a_1 \in \mathcal{A}_1} Q^*(s, a_1) & \text{if } s \in \mathcal{S}_1 \\ \min_{a_2 \in \mathcal{A}_2} Q^*(s, a_2) & \text{otherwise} \end{cases} \tag{3.2.4}$$

$$Q^*(s,a) = R(s,a) + \gamma \sum_{s'} P(s,a,s')V^*(s'). \tag{3.2.5}$$

3.3. Solving Combinatorial Optimization problems with GNN and RL

In this section, we review a procedure to heuristically solve classical combinatorial optimization problems involving a single agent on a graph using a combination of GNN and RL. In the next chapter, we tackle the multi-agent case. Here, the purpose is to learn a heuristic proposing good quality strategies to solve any instance sampled from a distribution $\mathbb{D}$ of instances of a given combinatorial optimization problem on a graph. Khalil et al. 2017 were the first to tackle such a task with a combination of a graph embedding method and reinforcement learning. The framework they propose, S2V-DQN, is general and can be applied to many different combinatorial problems over graphs were a single agent must take decisions at the node level (e.g., *the Minimum Vertex Cover (MVC) problem which seeks to find a minimum sized set of nodes that covers all the edges of the graph).* In their setting, all the nodes from a graph $G = (V,A)$ are embedded using the `structure2vec` [Dai et al., 2016] algorithm. A state s is described with the sequence of the previously taken actions, which correspond to a *partial solution* of the problem. The idea is to begin at $t = 0$ with $s = \emptyset$, and finish at $t = T$ with s a solution to the instance of the problem. The representation of s is simply the sum of the corresponding node embeddings. The transition is completely deterministic, the next state of the environment being obtained by tagging the node v corresponding to the action taken with the label $x_v = 1$. The action space at each time step thus corresponds to the set of free nodes at t *i.e.*$\{v \in V : x_v = 0\}$. The definition of the reward depends on the problem at hand, in the case of the MVC, each new action leads to a negative reward -1 and the procedure terminates when the nodes in the partial solution s constitute a proper cover. Their agent learns to estimate the Q values of all the state-action pairs possible to encounter in the distribution of instances $\mathbb{D}$ by using an adaptation of DQN, which they name S2V-DQN. Once their graph neural network is trained, they heuristically solve any new instance of the problem by incrementally building a solution following the greedy policy. Thanks to this meta-algorithm, Khalil et al. 2017 managed to show promising results on three classic budget-free NP-hard problems. Thenceforth, there is a growing number of methods proposed to either improve upon S2V-DQN results [Barrett et al., 2020; Cappart et al., 2019; Kool et al., 2019; Li et al., 2018; Ma et al., 2019] or tackle other types of NP-hard problems on graphs [Bai et al., 2020]. Although there are variations on the way nodes are embedded or on the RL algorithm used, all these recent methods rely on the same global idea of incrementally building a solution to an instance of a combinatorial problem by following a greedy policy given by a GNN trained on a distribution of instances; see [Mazyavkina et al., 2020] for a recent survey.

Chapter 4

Learning heuristics for Multilevel Budgeted Combinatorial Optimization problems

This chapter has appeared in a paper under review at the NeurIPS 2020 conference: "Curriculum learning for multilevel budgeted combinatorial problems" [Nabli and Carvalho, 2020] by Adel Nabli and Margarida Carvalho.

Contribution. *This paper is the result of an original idea of mine. I was involved in all aspects of this scientific work: literature review, theoretical results, writing and coding.*

Although the approaches presented in the last chapter show promising results on many fundamental NP-hard problems over graphs, such as Maximum Cut [Barrett et al., 2020] or the Traveling Salesman Problem [Kool et al., 2019], the range of combinatorial challenges on which they are directly applicable is still limited.

Indeed, most of the combinatorial problems over graphs solved heuristically with Deep Learning [Bai et al., 2020; Barrett et al., 2020; Bello et al., 2016; Cappart et al., 2019; Khalil et al., 2017; Kool et al., 2019; Li et al., 2018; Ma et al., 2019] are classic NP-hard problems for which the canonical optimization formulation is a single-level Mixed Integer Linear Program: there is one decision-maker[1] seeking to minimize a linear cost subject to linear constraints and integer requirements. However, in many real-world situations, decision-makers interact with each other. A particular case of such setting are sequential games with a hierarchy between players: an upper level authority (a leader) optimizes its goal subject to the response of a sequence of followers seeking to optimize their own

1. We will use interchangeably the words decision-maker, agent and player. Note that decision-maker, player and agent are usually used in Operations Research, Game Theory and Reinforcement Learning, respectively. Similarly for the words decision, strategy and policy.

objectives given the actions previously made by others higher in the hierarchy. These problems are naturally modeled as *Multilevel Programming problems* (MPs) and can be seen as a succession of nested optimization tasks, *i.e.* mathematical programs with optimization problems in the constraints [Bracken and McGill, 1973; Candler and Norton, 1977; Zhang et al., 2015].

Thus, finding an optimal strategy for the leader in the multilevel setting may be harder than for single-level problems as evaluating the cost of a given strategy might not be possible in polynomial time: it requires solving the followers optimization problems. In fact, even Multilevel Linear Programming with a sequence of $L+1$ players (levels) is Σ_L^p-hard [Blair, 1992; Dudás et al., 1998; Jeroslow, 1985]. In practice, exact methods capable to tackle medium-sized instances in reasonable time have been developed for max-min-max Trilevels, min-max Bilevels and more general Bilevel Programs (*e.g.*, [Carvalho et al., 2020; Fischetti et al., 2017, 2019; Lozano and Smith, 2017; Tahernejad et al., 2020]).

Despite the computational challenges intrinsic to MPs, these formulations are of practical interest as they properly model hierarchical decision problems. Originally appearing in economics in the bilevel form, designated as *Stackelberg competitions* [von Stackelberg, 1934], they have since been extended to more than two agents and seen their use explode in Operations Research [Lachhwani and Dwivedi, 2017; Sinha et al., 2018]. Thus, research efforts have been directed at finding good quality heuristics to solve those problems, *e.g.* [DeNegre, 2011; Fischetti et al., 2018; Forghani et al., 2020; Talbi, 2013]. Hence, one can ask whether we can make an agent learn how to solve a wide range of instances of a given multilevel problem, extending the success of recent Deep Learning approaches on solving single-level combinatorial problems to higher levels.

In this chapter, we propose a simple curriculum to learn to solve a common type of multilevel combinatorial optimization problem: budgeted ones that are zero-sum games played in a graph. Although the framework we devise is set to be general, we center our attention on the Multilevel Critical Node problem (MCN) [Baggio et al., 2020] and its variants.

Our contribution rests on several steps. First, we frame generic *Multilevel Budgeted Combinatorial problems* (MBC) as *Alternating Markov Games* [Littman, 1996; Littman and Szepesvari, 1996]. This allows us to devise a first algorithm, MultiL-DQN, to learn Q-values. By leveraging both the combinatorial setting *(the environment is deterministic)* and the budgeted case *(the length of an episode is known in advance)*, we motivate a curriculum, MultiL-Cur. Introducing a Graph Neural Networks based agent, we empirically demonstrate the efficiency of our curriculum on 3 versions of the MCN, reporting results close to optimality on graphs of size up to 100.

Section 4.1 formalizes the MBC problem. In Section 4.2, we provide an overview of the relevant literature. The MBC is formulated within the Multi-Agent RL setting in Section 4.3, which leads to the presentation of our algorithmic approaches: MultiL-DQN and MultiL-Cur in Section 4.4 and Section 4.5, respectively. Section 4.6 describes the neural network architecture we used to validate our methodology in Section 4.7.

4.1. Problem statement

The general setting for the MPs we are considering is the following: given a graph $G = (V, A)$, two concurrent players, the *leader* and the *follower*, compete over the same combinatorial quantity S, with the *leader* aiming to maximize it and the *follower* to minimize it. They are given a total number of moves $L \in \mathbb{N}$ and a sequence of budgets $(b_1, ..., b_L) \in \mathbb{N}^L$. Although our study and algorithms also apply to general integer cost functions c, for the sake of clarity, we will only consider situations where the cost of a move is its cardinality. We focus on perfect-information games, *i.e.* both players have full knowledge of the budgets allocated and previous moves. The *leader* always begins and the last move is attributed by the parity of L. At each turn $l \in [\![1, L]\!]$, the player concerned makes a set of b_l decisions about the graph. This set is denoted by A_l and constrained by the previous moves $(A_1, .., A_{l-1})$. We consider games where players can only improve their objective by taking a decision: there is no incentive to pass. Without loss of generality, we can assume that L is odd. Then, the *Multilevel Budgeted Combinatorial* problem (MBC) can be formalized as:

$$\text{(MBC)} \qquad \max_{|A_1| \leq b_1} \min_{|A_2| \leq b_2} ... \max_{|A_L| \leq b_L} S(G, A_1, A_2, ..., A_L). \tag{4.1.1}$$

MBC is a zero-sum game as both leader and follower have the same objective function but their direction of optimization is opposite. A particular combinatorial optimization problem is defined by specifying the quantity S, fixing L, and by characterizing the nature of both the graph *(e.g directed, weighted)* and of the actions allowed at each turn *(e.g labeling edges, removing nodes)*. The problem being fixed, a distribution $\mathbb{D}$ of instances $i \sim \mathbb{D}$ is determined by setting a sampling law for random graphs and for the other parameters, having specified bounds beforehand: $n = |V| \in [\![n^{min}, n^{max}]\!]$, $|A| \in [\![d^{min} \times n(n-1), d^{max} \times n(n-1)]\!]$, $(b_1, ..., b_L) \in [\![b_1^{min}, b_1^{max}]\!] \times ... \times [\![b_L^{min}, b_L^{max}]\!]$. Our ultimate goal is thus to learn good quality heuristics that manage to solve each $i \sim \mathbb{D}$.

In order to achieve that, we aim to leverage the recurrent structures appearing in the combinatorial objects in the distribution $\mathbb{D}$ by learning graph embeddings that could guide the decision process. As data is usually very scarce (datasets of exactly solved instances being hard to produce), the go-to framework to learn useful representations in these situations is Reinforcement Learning.

4.2. Related Work

The combination of graph embedding with reinforcement learning to learn to solve distributions of instances of combinatorial problems was discussed in Chapter 3. As all the approaches we mentioned focus on single player games, they are not directly applicable to MBC.

To tackle the multiplayer case, Multi-Agent Reinforcement Learning (MARL) appears as the natural toolbox [Littman, 1994; Shoham et al., 2007]. The combination of Deep Learning with RL recently led to one of the most significant breakthrough in perfect-information, sequential two-player games: AlphaGo [Silver et al., 2017]. Although neural network based agents managed to exceed human abilities on other combinatorial games (*e.g.* backgammon [Tesauro, 2002]), these approaches focus on one fixed board game. Thus, they effectively learn to solve only one (particularly hard) instance of a combinatorial problem, whereas we aim to solve a whole distribution of them. Hence, the MBC problem we propose to study is at a crossroads between previous works on MARL and deep learning for combinatorial optimization.

Finally, taking another direction, some shifted their attention from specific problems to rather focus on general purpose solvers. For example, methodologies have been proposed to speed up the branch-and-bound implementations for (single-level) linear combinatorial problems by learning to branch [Balcan et al., 2018] using Graph Convolutional Neural Networks [Gasse et al., 2019], Deep Neural Networks [Zarpellon et al., 2020] or RL [Etheve et al., 2020] to name some recent works; see the surveys [Bengio et al., 2018; Lodi and Zarpellon, 2017]. To the best of our knowledge, the literature on machine learning approaches for general multilevel optimization is restricted to the linear non-combinatorial case. For instance, in [He et al., 2014; Shih et al., 2004] the linear multilevel problems are converted into a system of differential equations and solved using recurrent neural networks.

4.3. MARL formulation of the Multilevel Budgeted Combinatorial problem

With the definition of the problem in Section 4.1 and the introduction to Alternating Markov Game of Section 3.2.2, we have all the elements to frame the MBC in this Markov Game framework. The *leader* is the maximizer and the *follower* the minimizer. The states

s_t consist of a graph G_t and a tuple of budgets $\mathcal{B}_t = (b_1^t, ..., b_L^t)$, beginning with $s_0 \sim \mathbb{D}$. Thus, the value function is defined with:

$$V^*(s_0) = \max_{|A_1| \leq b_1^0} \min_{|A_2| \leq b_2^0} ... \max_{|A_L| \leq b_L^0} S(G_0, A_1, A_2, ..., A_L). \tag{4.3.1}$$

The game is naturally sequential with an episode length of L: each time step t corresponds to a level $l \in [\![1,L]\!]$. The challenge of such formulation is the size of the action space that can become large quickly. Indeed, in a graph G with n nodes, and *leader*'s budget b_1, if the action that he/she can perform is *"removing a set of nodes from the graph"* (a common move in network interdiction games), then the size of the action space for the first move of the game is $\binom{n}{b_1}$. To remedy this, we define $\mathcal{A}_1, ..., \mathcal{A}_L$ the sets of *individual* decisions available at each level l. Then, we make the simplifying observation that a player making a *set* of b_l decisions A_l in one move is the same as him/her making a *sequence* of b_l decisions $(a_l^1, ..., a_l^{b_l}) \in \mathcal{A}_l \times (\mathcal{A}_l \backslash \{a_l^1\}) \times ... \times \left(\mathcal{A}_l \backslash \{a_l^1, .., a_l^{b_l-1}\}\right)$ in one strike. More formally, we have the simple lemma :

Lemma 4.3.1. *The Multilevel Budgeted Combinatorial optimization problem (4.1.1) is equivalent to:*

$$\max_{a_1^1 \in \mathcal{A}_1} ... \max_{a_1^{b_1} \in \mathcal{A}_1 \backslash \{a_1^1, .., a_1^{b_1-1}\}} \min_{a_2^1 \in \mathcal{A}_2} ... \max_{a_L^{b_L} \in \mathcal{A}_L \backslash \{a_L^1, .., a_L^{b_L-1}\}} S(G, \{a_1^1, .., a_1^{b_1}\}, .., \{a_L^1, .., a_L^{b_L}\}).$$

Proof. We immediately have the following relation:

$$\max_{|A_1| \leq b_1} ... \max_{|A_L| \leq b_L} S(G, A_1, A_2, .., A_L) = \max_{a_1^1 \in \mathcal{A}_1} \max_{|A'_1| \leq b_1 - 1} ... \max_{|A_L| \leq b_L} S(G, \{a_1\} \cup A'_1, A_2, .., A_L)$$

As the same reasoning holds with min, we can apply it recursively, which closes the proof. □

In this setting, the length of an episode is no longer L but $B = b_1 + ... + b_L$: the *leader* makes b_1 sequential actions, then the *follower* the b_2 following ones, etc. To simplify the notations, we re-define the $\mathcal{A}_t$ as the sets of actions available for the agent playing at time t. As each action takes place on the graph, $\mathcal{A}_t$ is readable from s_t. Moreover, we now have $|\mathcal{A}_t| = \mathcal{O}(|V| + |A|)$.

The environments considered in MBC are deterministic and their dynamics are completely known. Indeed, given a graph G_t, a tuple of budgets $\mathcal{B}_t = (b_1^t, .., b_L^t)$ and a chosen action $a_t \in \mathcal{A}_t$ *(e.g removing the node a_t)*, the subsequent graph G_{t+1}, tuple of budgets $\mathcal{B}_{t+1}$ and next player are completely and uniquely determined. Thus, we can introduce *the next state function N* that maps state-action couples (s_t, a_t) to the resulting afterstate s_{t+1}, and p as the function that maps the current state s_t to the player $p \in \{1,2\}$ whose turn it is to play. As early rewards weight the same as late ones, we set $\gamma = 1$. Finally, we can re-write equations (3.2.4) and (3.2.5) as:

$$V^*(s_t) = \begin{cases} \max_{a_t \in \mathcal{A}_t} \left(R(s_t, a_t) + V^*(N(s_t, a_t))\right) & \text{if } p(s_t) = 1 \\ \min_{a_t \in \mathcal{A}_t} \left(R(s_t, a_t) + V^*(N(s_t, a_t))\right) & \text{otherwise} \end{cases} \tag{4.3.2}$$

$$Q^*(s_t, a_t) = R(s_t, a_t) + V^*(N(s_t, a_t)). \tag{4.3.3}$$

The definition of R depends on the combinatorial quantity S and the nature of the actions allowed.

4.4. Q-learning for the greedy policy

Having framed the MBC in the Markov Game framework, the next step is to look at established algorithms to learn Q^* in this setup. Littman originaly presented *minimax Q-learning* [Littman, 1994, 1996] to do so, but in matrix games, where all possible outcomes are enumerated. An extension using a neural network $\hat{Q}$ to estimate Q has been discussed in [Fan et al., 2019]. However, their algorithm, Minimax-DQN, is suited for the simultaneous setting and not the alternating one. The main difference being that the former requires the extra work of solving a Nash game between the two players at each step, which is unnecessary in the later as a greedy policy exists [Littman, 1994]. To bring Minimax-DQN to the alternating case, we present MultiL-DQN (Algorithm 1), an algorithm inspired by S2V-DQN [Khalil et al., 2017] but extended to the multilevel setting. Compared to S2V-DQN, there is an alternation between the use of "min" and "max" in the greedy rollout, as well as in the target definition. As the player currently playing is completely determined from s_t, we can use the same neural network $\hat{Q}$ to estimate all the state-action values, regardless of the player. We call B_t the sum of all the budgets in $\mathcal{B}_t$ such that an episode stops when $B_t = 0$.

4.5. A curriculum taking advantage of the budgeted combinatorial setting

With MultiL-DQN, the learning agent directly tries to solve instances drawn from $\mathbb{D}$, which can be very hard theoretically speaking. However, Lemma 4.3.1 shows that, at the finest level of granularity, MBC is actually made of B nested sub-problems. As we know $B_{max} = b_1^{max} + ... + b_L^{max}$, the maximum number of levels considered in $\mathbb{D}$, instead of directly trying to learn the values of the instances from this distribution, we can ask whether beginning by focusing on the simpler sub-problems and gradually build our way up to the hard ones would result in better final results.

This reasoning is motivated by the work done by [Bengio et al., 2009] on Curriculum Learning. Indeed, it has been shown empirically that breaking down the target training

Algorithm 1: MultiL-DQN

```
1  Initialize the replay memory $\mathcal{M}$ to capacity $\mathcal{C}$ ;
2  Initialize the $Q$-network $\hat{Q}$ with weights $\hat{\theta}$ ;
3  Initialize the target-network $\tilde{Q}$ with weights $\tilde{\theta} = \hat{\theta}$ ;
4  for episode $e = 1, ..., E$ do
5      Sample $s_0 = (G_0, \mathcal{B}_0) \sim \mathbb{D}$ ;
6      $t \leftarrow 0$ ;
7      while $B_t \geq 1$ do
8          $a_t = \begin{cases} \text{random action } a_t \in \mathcal{A}_t & \text{w.p. } \epsilon \\ \arg\max_{a_t \in \mathcal{A}_t} \hat{Q}(s_t, a_t) & \text{otherwise if } p(s_t) = 1 \\ \arg\min_{a_t \in \mathcal{A}_t} \hat{Q}(s_t, a_t) & \text{otherwise if } p(s_t) = 2 \end{cases}$ ;
9          $s_{t+1} \leftarrow N(s_t, a_t)$ ;
10         $t \leftarrow t + 1$ ;
11         if $t \geq 1$ then
12             Add $(s_{t-1}, a_{t-1}, R(s_{t-1}, a_{t-1}), s_t)$ to $\mathcal{M}$ ;
13             Sample a random batch $\{(s_i, a_i, r_i, s'_i)\}_{i=1}^m \overset{i.i.d}{\sim} \mathcal{M}$ ;
14             for $i = 1,..,m$ do
15                 $y_i = r_i + \mathbb{1}_{p(s'_i)=1} \max_{a' \in \mathcal{A}'} \tilde{Q}(s'_i, a') + \mathbb{1}_{p(s'_i)=2} \min_{a' \in \mathcal{A}'} \tilde{Q}(s'_i, a')$
16             Update $\hat{\theta}$ over $\frac{1}{m} \sum_{i=1}^m \left(y_i - \hat{Q}(s_i, a_i)\right)^2$ with Adam [Kingma and Ba, 2015];
17             Update $\tilde{\theta} \leftarrow \hat{\theta}$ every $T_{target}$ steps
18 return the trained $Q$-network $\hat{Q}$
```

distribution into a sequence of increasingly harder ones actually results in better generalization abilities for the learning agent. But, contrary to the *continuous* setting devised in their work, where the parameter governing the hardness *(entropy)* of the distributions considered is continuously varying between 0 and 1, here we have a natural *discrete* sequence of increasingly harder distributions to sample instances from.

Indeed, our ultimate goal is to learn an approximate function $\hat{Q}$ to Q^* *(or equivalently $\hat{V}$ to V^* (4.3.3))* so that we can apply the subsequent greedy policy to take a decision. Thus, $\hat{Q}$ has to estimate the state-action values of every instance appearing in a sequence of B decisions. Although the *leader* makes the first move on instances from $\mathbb{D}$, as our game is played on the graph itself, the distribution of instances on which the second decision is made is no longer $\mathbb{D}$ but *instances from $\mathbb{D}$ on which a first optimal action for the leader has been made.* If we introduce the function

$$\begin{aligned} N^k_{l,\pi^*} : \quad & \mathcal{S} \longrightarrow \mathcal{S} \\ s_t = (G_t, \mathcal{B}_t) \quad & \mapsto \begin{cases} N(s_t, a^*_t \sim \pi^*(\mathcal{A}_t)) & \text{if} \quad \mathcal{B}_t = (0,..,0,k, b_{l+1}, .., b_L) \\ s_t & \text{otherwise} \end{cases} \end{aligned} \tag{4.5.1}$$

then, from top to bottom, we want $\hat{Q}$ to estimate the values of taking an action starting from the states $\mathbb{D}^*_1 = \mathbb{D}$ where the first action is made, then $\mathbb{D}^*_2 = N^{b_1^{max}}_{1,\pi^*}(\mathbb{D})$ where the second action is made on instances with original budget b_1^{max} at level 1, and all the way down to

$$\mathbb{D}^*_{B_{max}} = N^2_{L,\pi^*} \circ ... \circ N^{b_L^{max}}_{L,\pi^*} \circ ... \circ N^{b_1^{max}}_{1,\pi^*}(\mathbb{D}). \tag{4.5.2}$$

As the maximum total budget is B_{max}, $\hat{Q}$ has to effectively learn to estimate values from B_{max} different distributions of instances, one for each possible budget in $[\![1, b_l^{max}]\!]$ for each $l \in [\![1, L]\!]$. But the instances in these distributions are not all equally hard to solve. Actually, the tendency is that the deeper in the sequence of decisions a distribution is, the easier to solve are the instances sampled from it. For example, the last distribution $\mathbb{D}^*_{B_{max}}$ contains all the instances with a total remaining budget of at most 1 that it is possible to obtain for the last move of the game when every previous action was optimal. The values of these instances can be computed exactly in polynomial time[2] by checking the reward obtained with every possible action. Thus, if we had access to the $\{D^*_j\}_{j=1}^{B_{max}}$, then a natural curriculum would be to begin by sampling a dataset of instances $s^*_{B_{max}} \sim \mathbb{D}^*_{B_{max}}$, find their exact value in polynomial time, and train a neural network $\hat{V}$ on the couples $(s^*_{B_{max}}, V^*(s^*_{B_{max}}))$. Once this is done, we could pass to the $s^*_{B_{max}-1} \sim \mathbb{D}^*_{B_{max}-1} = N^3_{L,\pi^*} \circ ... \circ N^{b_1^{max}}_{1,\pi^*}(\mathbb{D})$. As these instances have a total budget of at most 2, we can heuristically solve them by generating every possible afterstate and, by using the freshly trained $\hat{V}$, take a greedy decision to obtain their approximate targets. In a bottom up approach, we could continue until $\hat{V}$ is trained on the B_{max} different distributions. The challenge of this setting being that we do not know π^* and hence, $\{\mathbb{D}^*_j\}_{j=1}^{B_{max}}$ are not available. To remedy this, we use a proxy, $\mathbb{D}^r_j$, obtained by following the random policy $a^r_t \sim \mathcal{U}(\mathcal{A}_t)$ for the sequence of previous moves, *i.e.*, we use N^k_{l,π^r} instead of N^k_{l,π^*}. Doing so is provably interesting:

Lemma 4.5.1. $\forall j \in [\![2, B_{max}]\!]$, supp$(\mathbb{D}^*_j) \subseteq$ supp$(\mathbb{D}^r_j)$.

PROOF. For all $s_0 \sim \mathbb{D}$, for all $t \in [\![0, B-1]\!]$, we define $\mathcal{A}^*_t(a_0, ..., a_{t-1}) \subseteq \mathcal{A}_t(a_0,...,a_{t-1})$ as the set of optimal actions at time t in state s_t for the player $p(s_t)$, where we made evident the dependence of s_t on previous actions. As by assumption we consider games where players can only improve their objective by taking a decision, we have that $\forall t$, $\mathcal{A}_t \neq \emptyset \implies \mathcal{A}^*_t \neq \emptyset$. For a given s_t and subsequent $\mathcal{A}_t$, recall that a^r_t is defined as a random variable with values in $\mathcal{A}_t$ and following the uniform law. Given $s_0 \sim \mathbb{D}$, we take $(a^*_0, ..., a^*_{B-1}) \in \mathcal{A}^*_0 \times ... \times \mathcal{A}^*_{B-1}(a^*_0,...,a^*_{B-2})$, one of the possible sequence of optimal decisions. Then, using the chain rule, it is easy to show by recurrence that $\forall t \in [\![0, B-1]\!]$, $P(a^r_0 = a^*_0, ..., a^r_t = a^*_t) > 0$. In words, every optimal sequence of decisions is generated with a strictly positive probability. □

2. Assuming the quantity S is computable in polynomial time.

Thus, by learning the value of instances sampled from $\mathbb{D}^r_j$, we also learn values of instances from $\mathbb{D}^*_j$. To avoid the pitfall of catastrophic forgetting [Lange et al., 2019] happening when a neural network switches of training distribution, each time it finishes to learn from a $\mathbb{D}^r_j$ and before the transition j to $j-1$, we freeze a copy of $\hat{V}$ and save it in memory as an *"expert of level j"*. All this leads to the algorithm MultiL-Cur:

Algorithm 2: MultiL-Cur

```
1  Initialize the value-network $\hat{V}$ with weights $\hat{\theta}$ ;
2  Initialize the list of experts $\mathbb{L}_{\hat{V}}$ to be empty ;
3  for $j = B_{max}, ..., 2$ do
4      Create $\mathcal{D}^j_{train}$, $\mathcal{D}^j_{val}$ by sampling $(s^r_j \sim \mathbb{D}^r_j$, GreedyRollout $(s^r_j, \mathbb{L}_{\hat{V}}))$;
5      Initialize $\hat{V}_j$, the expert of level $j$ with $\hat{\theta}_j = \hat{\theta}$ ;
6      Initialize the loss on the validation set $\mathcal{L}^j_{val}$ to $\infty$ ;
7      for epoch $e = 1,...,E$ do
8          for batches $(s_i, \hat{y}_i)^m_{i=1} \in \mathcal{D}^j_{train}$ do
9              Update $\hat{\theta}$ over $\frac{1}{m}\sum_{i=1}^m \left(\hat{y}_i - \hat{V}(s_i)\right)^2$ with Adam [Kingma and Ba, 2015];
10             if number of new updates $= T_{val}$ then
11                 if $\mathcal{L}^{new}_{val} = \frac{1}{N_{val}}\sum_{k=1}^{N_{val}} \left(\hat{y}_k - \hat{V}(s_k)\right)^2 < \mathcal{L}^j_{val}$ then
12                     $\hat{\theta}_j \leftarrow \hat{\theta}$ ; $\mathcal{L}^j_{val} \leftarrow \mathcal{L}^{new}_{val}$ ;
13     Add $\hat{V}_j$ to $\mathbb{L}_{\hat{V}}$
14 return the trained list of experts $\mathbb{L}_{\hat{V}}$
```

In order to find the approximate target of an instance $s^r_j \sim \mathbb{D}^r_j$, MultiL-Cur takes a sequence of greedy decisions using the list of previously trained experts $\mathbb{L}_{\hat{V}}$ until all the budget in s^r_j is spent. The procedure is summarized in the Greedy Rollout algorithm:

Algorithm 3: Greedy Rollout

```
   Input: A state $s_t$ with total budget $B_t$ and a list of experts value networks $\mathbb{L}_{\hat{V}}$
1  Initialize the value $\hat{v} \leftarrow 0$ ;
2  while $B_t \geq 1$ do
3      Retrieve the expert of the next level $\hat{V}_{t+1}$ from the list $\mathbb{L}_{\hat{V}}$ ;
4      Generate every possible afterstate $\mathcal{S}'_t \leftarrow \{N(s_t, a_t)\}_{a_t \in \mathcal{A}_t}$ ;
5      $s_{t+1} = \begin{cases} \arg\max_{s' \in \mathcal{S}'_t} \hat{V}_{t+1}(s') & \text{if } p(s_t) = 1 \\ \arg\min_{s' \in \mathcal{S}'_t} \hat{V}_{t+1}(s') & \text{if } p(s_t) = 2 \end{cases}$ ;
6      $\hat{v} \leftarrow \hat{v} + R(s_t, s_{t+1})$ ;
7      $t \leftarrow t + 1$ ;
8  return the value $\hat{v}$
```

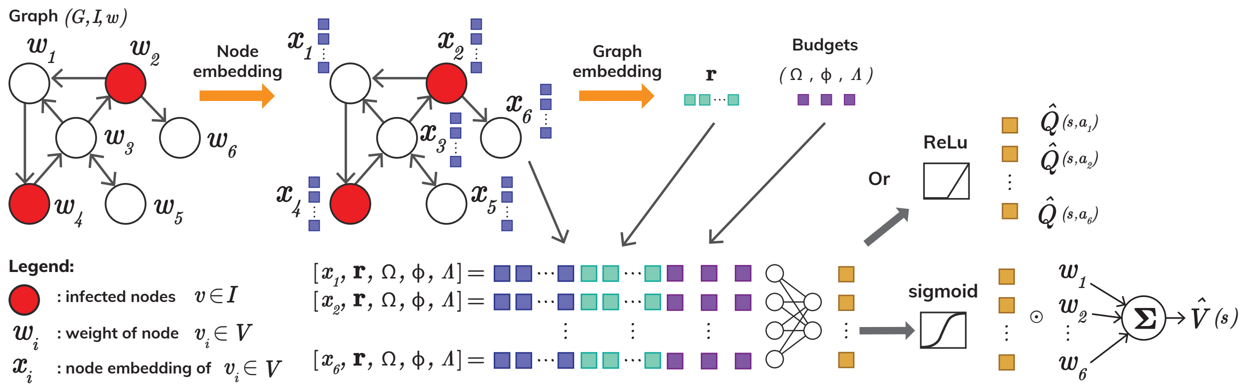

Graph (G,I,w)
Node embedding
Graph embedding
Budgets
(Ω, ϕ, Λ)
$\mathbf{r}$
ReLu
Or
sigmoid
Legend:
: infected nodes $v \in I$
w_i : weight of node $v_i \in V$
x_i : node embedding of $v_i \in V$
$[x_1, \mathbf{r}, \Omega, \phi, \Lambda] =$
$[x_2, \mathbf{r}, \Omega, \phi, \Lambda] =$
$[x_6, \mathbf{r}, \Omega, \phi, \Lambda] =$
$\hat{Q}(s,a_1)$
$\hat{Q}(s,a_2)$
$\hat{Q}(s,a_6)$
$\hat{V}(s)$

Figure 4.1. Architecture of the two neural networks used to solve the MCN: $\hat{V}$ and $\hat{Q}$. $\hat{V}$ computes a score $\in$ [0,1] for each node, which can be interpreted as its probability to be saved given the context (graph embedding and budgets).

4.6. GNN architecture for the MCN problem

The problem we focused on being the MCN, we designed a GNN fit to the problem, see Figure 4.1 for an overview of the architecture. We implemented it with Pytorch Geometric [Fey and Lenssen, 2019] and Pytorch 1.4 [Paszke et al., 2019]. To train our agent and at inference, we used one gpu of a cluster of NVidia V100SXM2 with 16G of memory [3].

4.6.1. Node embedding

The first step of the method described in Figure 4.1 is the node embedding part. Each node $v \in V$ begins with two features $\mathbf{x_v} = (w_v, \mathbb{1}_{v \in I})$: its weight and an indicator of whether it is attacked or not. First, we normalize the weights by dividing them with the sum of the weights in the graph such that each $w_v \in$ [0,1]. We extend the two features with the *Local Degree Profile* of each node [Cai and Wang, 2018], which consists in 5 features on the degree:

$$\mathbf{x_v} = \mathbf{x_v} || (\deg(v), \min(DN(v)), \max(DN(v)), \mathrm{mean}(DN(v)), \mathrm{std}(DN(v))) \tag{4.6.1}$$

with deg the degree of a node, and $DN(v)$ the vector of the degrees of $\mathcal{N}(v)$. Then, we project our features $\mathbf{x_v} \in \mathbb{R}^7$ into $\mathbb{R}^{d_e}$ with a linear layer. Following the success of Attention on routing problems reported in [Kool et al., 2019], we then apply their Multihead Attention Layer using a Graph Attention Network (GAT) [Velicković et al., 2018]. Thus, we apply one GAT layer such that:

$$\mathbf{x_v}' = \mu_{v,v} \mathbf{\Theta} \mathbf{x_v} + \sum_{u \in \mathcal{N}(v)} \mu_{v,u} \mathbf{\Theta} \mathbf{x_u} \tag{4.6.2}$$

with μ defined by:

$$\mu_{v,u} = \frac{\exp(\mathrm{LeakyReLU}(\mathbf{a}^\top [\mathbf{\Theta x_v} || \mathbf{\Theta x_u}]))}{\sum_{k \in \mathcal{N}(v) \cup \{v\}} \exp(\mathrm{LeakyReLU}(\mathbf{a}^\top [\mathbf{\Theta x_v} || \mathbf{\Theta x_k}]))}, \tag{4.6.3}$$

3. We make our code publicly available: `https://github.com/AdelNabli/MCN`

where $\mathbf{a} \in \mathbb{R}^{2\times d_v}$ and $\mathbf{\Theta} \in \mathbb{R}^{d_v \times d_e}$ are the trainable parameters. Here, d_e is the original embedding dimension of $\mathbf{x_v}$, d_v is the dimension of $\mathbf{x_v}'$. We apply these equations with n_h different $\mathbf{\Theta}$ and $\mathbf{a}$, n_h being the number of heads used in the attention layer. Then, we project back in $\mathbb{R}^{d_e}$ the $\mathbf{x_v}'$ with a linear layer, and sum the n_h resulting vectors. After that, we apply a skip connection [He et al., 2016] and a Batch-Normalization [Ioffe and Szegedy, 2015] layer BN such that:

$$\mathbf{x_v}' = \text{BN}(\mathbf{x_v} + \mathbf{x_v}'). \tag{4.6.4}$$

Finally, we introduce a feedforward network FF which is a 2-layer fully connected network with ReLU activation functions. The input and output dimensions are d_e and the hidden dimension is d_h. The final output is then:

$$\mathbf{x_v} = \text{BN}(\mathbf{x_v}' + \text{FF}(\mathbf{x_v}')). \tag{4.6.5}$$

We repeated the process described between Equation (4.6.2) and Equation (4.6.5) a total of n_a times. As infected nodes are the ones in the same connected component as attacked ones in the graph, we sought to propagate the information of each node to all the others it is connected to. That way, the *attacker* could know which nodes are already infected before spending the rest of his/her budget, and the *defender* could realize which nodes are to protect in his/her last move. So, after the Attention Layers, we used an APPNP layer [Klicpera et al., 2019] that, given the matrix of nodes embedding $\mathbf{X}^{(0)}$, the adjacency matrix with inserted self-loops $\hat{\mathbf{A}}$, $\hat{\mathbf{D}}$ its corresponding diagonal degree matrix, and a coefficient $\alpha \in [0,1]$, recursively applies K times:

$$\mathbf{X}^{(k)} = (1-\alpha)\hat{\mathbf{D}}^{-1/2}\hat{\mathbf{A}}\hat{\mathbf{D}}^{-1/2}\mathbf{X}^{(k-1)} + \alpha\mathbf{X}^{(0)}. \tag{4.6.6}$$

To achieve our goal, the value of K must be at least equal to the size of the largest connected component possible to have in the distribution of instances $\mathbb{D}$ we are considering, we thus set K to be equal to the largest number of nodes a graph could have in $\mathbb{D}$.

4.6.2. Graph embedding

Given the resulting nodes embedding $\mathbf{x_v} \in \mathbb{R}^{d_e}$, in a skip-connection fashion, we concatenate the $\mathbf{x_v}$ back with the original two features $(w'_v, \mathbb{1}_{v\in I})$ *(w'_v being the normalized weights)*. Finally, the graph embedding method we used is the one presented in [Li et al., 2016]. Given two neural networks h_{gate} and h_r which compute, respectively, a score $\in \mathbb{R}$ and a projection to $\mathbb{R}^r$, the graph level representation vector is, for a graph of size n:

$$\mathbf{r} = \sum_{i=1}^{n} \text{softmax}(h_{gate}(\mathbf{x_i})) \odot h_r(\mathbf{x_i}) \tag{4.6.7}$$

where $\odot$ denotes the Hadamard product. Here, h_{gate} and h_r are feedforward neural networks with 2 layers and using ReLU activation functions. For both, the input dimension is $d_e + 2$

and the hidden dimension is d_h. For h_r the output dimension is d_e whereas for h_{gate}, it is 1. We used n_p different versions of the parameters and concatenated the n_p different outputs such that the final graph embedding has a dimension of $n_p \times d_e$.

4.6.3. Final steps

We now have the nodes embedding $\mathbf{x_v} \in \mathbb{R}^{d_e}$ and a graph representation $\mathbf{r}$ of dimension $d_e \times n_p$. But the context for each node is not entirely contained in $\mathbf{r}$: the budgets, the size of the graph n and the total sum of weights in the graph are still missing. Thus, we form a context vector c_o as follows:

$$c_o = \mathbf{r}||(n, \Omega, \Phi, \Lambda, \Omega/n, \Phi/n, \Lambda/n, \sum_{v \in V} w_v). \tag{4.6.8}$$

When this is done, we perform, for each node, the concatenation $\mathbf{x_v}||c_o$. This is the entry of a feedforward neural network, FF_V or FF_Q, that computes, for $\hat{V}$, the probability of each node being saved given the context, and the state-action values for $\hat{Q}$. The two feedforward networks are 3-layers deep, with the first hidden dimension being d_h and the second d_e. We used LeakyReLU activation functions, Batch Norm and dropout [Srivastava et al., 2014] with parameter p. Indeed, our experiment shows that using dropout at this stage helps prevent overfitting, and Batch Norm speedups the training. The last activation function for FF_Q is ReLU whereas for FF_V we use a sigmoid. Finally, for FF_V, we output:

$$\hat{V}(s) = \sum_{v \in V} P(v \text{ is saved} \mid \text{context}) w_v. \tag{4.6.9}$$

For FF_Q, we just mask the actions not available, *i.e.* the nodes that are already labeled as attacked.

4.6.4. Hyperparameters

All the negative slopes in the LeakyReLU we used were set by default at 0.2. The value of all the other hyperparameters we introduced here were fixed using Optuna [Akiba et al., 2019] with a TPE sampler and a Median pruner. The objective we defined was the value of the loss of $\hat{V}$ on a test set of exactly solved instances with budgets $\Omega = 0,\ \Phi = 1,\ \Lambda \in [\![0,3]\!]$. After running Optuna for 100 trials, we fixed the following values for the hyperparameters: $d_e = 200,\ d_h = 400,\ d_v = 100,\ \alpha = 0.2,\ p = 0.2,\ n_a = 7,\ n_h = 3,\ n_p = 3$. It represents a total of 2,8 million parameters to train for both $\hat{V}$ and $\hat{Q}$.

4.7. Computational results

In order to test the validity of the approaches we discussed in this chapter, we tackled several versions of the MCN. In this section, we report in details the computational results we obtained while training our GNN agents using the algorithms derived earlier. First, we detail the distributions of instances we considered in Section 4.7.1. Then, in Section 4.7.2, we report results of the comparison between the different approaches. Finally, in Section 4.7.3, we explore in more details the abilities of our best performing algorithm, MultiL-Cur.

4.7.1. Distribution of instances

We studied 3 versions of the MCN: undirected with unit weights (MCN), undirected with positive weights (MCN$_w$), and directed with unit weights (MCN$_{dir}$). The first distribution of instances considered is $\mathbb{D}^{(1)}$, constituted of Erdos-Renyi graphs [Erdos and Renyi, 1960] with size $|V|^{(1)} \in [\![10,23]\!]$ and arc density $d^{(1)} \in [0.1,0.2]$. For the weighted case, we considered integer weights $w \in [\![1,5]\!]$. The second distribution of instances $\mathbb{D}^{(2)}$ focused on larger graphs with $|V|^{(2)} \in [\![20,60]\!]$, $d^{(2)} \in [0.05,0.15]$. To compare our results with exact ones, we used the budgets reported in the experiments of the original MCN paper [Baggio et al., 2020]: $\Omega \in [\![0,3]\!]$, $\Phi \in [\![1,3]\!]$ and $\Lambda \in [\![0,3]\!]$. We compared our algorithms on $\mathbb{D}^{(1)}$ (Section 4.7.2) and trained our best performing one on $\mathbb{D}^{(2)}$ (Section 4.7.3).

4.7.2. Comparison between the different algorithms

In order to properly compare our methods, first, we introduce in Section 4.7.2.1 a third algorithm to resolve some of the handicaps MultiL-DQN has compared to MultiL-Cur. Then, to be able to quantify the quality of our procedures, we discuss in Section 4.7.2.2 how to adapt the methods from [Baggio et al., 2020] to exactly solve instances from each of the 3 versions of the MCN we studied. Then, we compare the algorithms in Section 4.7.2.3 and contrast their sensitivity to the abundance of data during training in Sections 4.7.2.4 and 4.7.2.5.

4.7.2.1. Introducing a third algorithm

As it is, comparing MultiL-DQN with MultiL-Cur may be unfair. Indeed, MultiL-DQN uses a Q-network whereas MultiL-Cur uses a value network. The reason why we used $\hat{V}$ instead of $\hat{Q}$ in our second algorithm are twofold. First, as our curriculum leans on the abilities of experts trained on smaller budgets to create the next training dataset, computing *values* of afterstates is necessary to heuristicaly solve instances with larger budgets. Second, as MCN is a game with one player removing nodes from the graph, symmetries can be leveraged in the afterstates. Indeed, given the graph G' resulting of a node deletion, many couples (G,v) of graph and node to delete could have resulted in G'. Thus, $\hat{Q}$ has to learn that all these possibilities are similar, while $\hat{V}$ only needs to learn the value of the shared afterstate, which is more efficient [Sutton and Barto, 1998]. To fairly compare the algorithms, we thus introduce MultiL-MC, a version of MultiL-DQN based on a value network and using Monte-Carlo samples as in MultiL-Cur.

```
Algorithm 4: MultiL-MC
1  Initialize the replay memory $\mathcal{M}$ to capacity $\mathcal{C}$ ;
2  Initialize the value-network $\hat{V}$ with weights $\hat{\theta}$ ;
3  for episode $e = 1, ..., E$ do
4      Sample $s_0 = (G_0, \mathcal{B}_0) \sim \mathbb{D}$ ;
5      Initialize the memory of the episode $\mathcal{M}_e$ to be empty;
6      Initialize the length of the episode $T \leftarrow 0$ ;
7      while $B_t \geq 1$ do                                  // perform a Monte Carlo sample
8          $a_t = \begin{cases} \text{random action } a_t \in \mathcal{A}_t & \text{w.p. } \epsilon \\ \arg\max_{a_t \in \mathcal{A}_t} \hat{V}(N(s_t, a_t)) & \text{otherwise if } p(s_t) = 1 \\ \arg\min_{a_t \in \mathcal{A}_t} \hat{V}(N(s_t, a_t)) & \text{otherwise if } p(s_t) = 2 \end{cases}$ ;
9          $s_{t+1} = N(s_t, a_t)$ ;
10         Add $(s_t, R(s_t, a_t))$ to $\mathcal{M}_e$ ;
11         $T \leftarrow T + 1$
12     Initialize the target $y_T \leftarrow 0$ ;
13     for $t = 1,...,T$ do                             // associate each state to its value
14         Recover $(s_{T-t}, R(s_{T-t}, a_{T-t}))$ from $\mathcal{M}_e$ ;
15         $y_{T-t} \leftarrow y_{T-t+1} + R(s_{T-t}, a_{T-t})$ ;
16         Add $(s_{T-t}, y_{T-t})$ to $\mathcal{M}$
17     if there are more than $m$ new couples in $\mathcal{M}$ then
18         Create a random permutation $\sigma \in \mathcal{S}_N$ ;
19         for batches $\{(s_i, y_i)\}_{i=1}^m \sim \sigma(\mathcal{M})$ do   // perform an epoch on the memory
20             Update $\hat{\theta}$ over the loss $\frac{1}{m}\sum_{i=1}^m \left(y_i - \hat{V}(s_i)\right)^2$ with Adam [Kingma and Ba,
               2015]
21 return the trained value-network $\hat{V}$
```

As we use Monte-Carlo samples as targets, the values of the targets sampled from the replay memory $\mathcal{M}$ is not dependent on the current expert as in DQN [Mnih et al., 2015] but on a previous version of $\hat{V}$, which can become outdated quickly. Thus, to easily control the number of times an old estimate is used, we decided to perform an epoch on the memory every time m new samples were pushed, and used a capacity $\mathcal{C} = k \times m$ so that the total number of times a Monte-Carlo sample is seen is directly k.

4.7.2.2. Broadening the scope of the exact algorithm

In order to constitute a test set to compare the results given by our heuristics to exact ones, we used the exact method described in [Baggio et al., 2020] to solve a small amount of instances. However, in order to monitor the learning at each stage of the curriculum, there is a need to solve instances where node infections were already performed in the sequence of previous moves but there is still some budget left to spend for the attacker, which is not possible as it is in [Baggio et al., 2020]. Moreover, small changes need to be made in order to solve instances of MCN$_w$. We thus added some small modification to the procedure of [Baggio et al., 2020] described in Section 2.2.

Adding nodes that are already infected. We denote by J the set of nodes that are already infected at the attack stage and $\beta_v = \mathbb{1}_{v \in J}$ the indicator of whether node v is in J or not. Then, the total set of infected nodes after the attacker spend his/her remaining budget Λ and infect new nodes I is $J \cup I$. In order to find I, we use the AP algorithm of [Baggio et al., 2020], with the following modification to the rlxAP optimization problem:

$$
\begin{aligned}
\min \quad & \Lambda p + \sum_{v \in V} \gamma_v & \\
& \sum_{v \in V} y_v \leq \Lambda & \\
& y_v \leq 1 - \beta_v & \forall v \in V \\
& h_v + \sum_{(u,v) \in A} q(u,v) - \sum_{(u,v) \in A} q(v,u) \geq 1 & \forall v \in V \\
& p - \sum_{(u,v) \in A} q(u,v) \geq 0 & \forall v \in V \\
& \gamma_v + |V| y_v - h_v \geq -|V|\beta_v & \forall v \in V \\
& p,\ h_v,\ \gamma_v,\ q(u,v) \geq 0 & \forall v \in V,\ \forall (u,v) \in A \\
& y_v \in \{0,1\} & \forall v \in V
\end{aligned}
$$

We indicated changes in blue. The notations for the variables being the ones from [Baggio et al., 2020].

Adding weights. Taking the weights w_v of the nodes $v \in V$ into account in the optimization problems is even more straightforward. As the criterion to optimize is no longer the *number* of saved nodes but the *sum of their weights*, each time a cardinal of a set appears in the algorithms AP and MCN in [Baggio et al., 2020], we replace it by the the sum of the weights of its elements. As for the optimization problems that are solved during the routines, we replace, in the Defender problem and in the 1lvlMIP:

$$\sum_{v \in V} \alpha_v \longrightarrow \sum_{v \in V} w_v \alpha_v$$

and in the rlxAP problem:

$$h_v + \sum_{(u,v)\in A} q(u,v) - \sum_{(u,v)\in A} q(v,u) \geq 1 \longrightarrow h_v + \sum_{(u,v)\in A} q(u,v) - \sum_{(u,v)\in A} q(v,u) \geq w_v$$

4.7.2.3. Comparison results

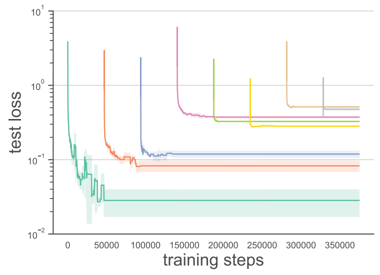
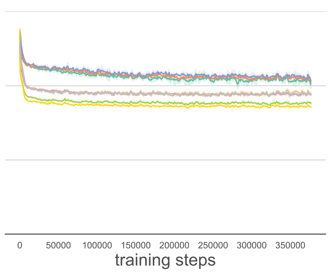
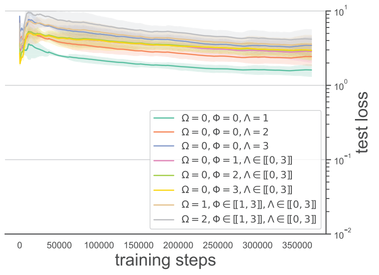
MultiL-Cur		MultiL-MC		MultiL-DQN	
					
opt gap (%)	**approx ratio**	**opt gap** (%)	**approx ratio**	**opt gap** (%)	**approx ratio**
0.51	**1.006**	3.34	1.036	14.53	1.158

Table 4.1. Evolution during training of the loss on 8 test sets of 1000 exactly solved instances $\in \mathbb{D}^{(1)}$. Averaged on 3 runs. We measured the loss on distributions arriving at different stages of the curriculum. The approximation ratio and optimality gap were measured after training and averaged over all the tests sets.

Results from Table 4.1 indicate that MultiL-Cur is the best performing algorithm on $\mathbb{D}^{(1)}$. The metrics we use are the optimality gap η and the approximation ratio ζ. Given n_i, the number of instances of a said type for which the optimal value v^* is available, $\eta = \frac{1}{n_i}\sum_{k=1}^{n_i}\frac{|v_k^* - \hat{v}_k|}{v_k^*}$ and $\zeta = \frac{1}{n_i}\sum_{k=1}^{n_i}\max(\frac{v_k^*}{\hat{v}_k}, \frac{\hat{v}_k}{v_k^*})$.

For all 3 algorithms, we used roughly the same values of parameters in order to make the comparison fair. All three algorithms were compared on instances from $\mathbb{D}^{(1)}$. The batch size was fixed to $m = 256$. Although we share our training times for the sake of transparency and to compare the methods, we want to highlight that our code is hardly optimized and that cutting the times presented here may be easy with a few improvements.

For MultiL-Cur, we used a training set of size 100 000 and a validation set of 1000 instances at each stage of the curriculum. As there are 8 distributions to learn from *(as we use afterstates, there is no need to learn the values of instances having* $\Omega = 3,\ \Phi = 3,\ \Lambda = 3$ *as budgets)*, this amounts for a total of 808 000 episodes. At each stage j, we trained our expert $\hat{V}_j$ for 120 epochs, meaning that we used a total of 375 000 training steps to finish the curriculum, which necessitated a total of 36 hours. Most of the training time was directed towards generating the training sets, *i.e.* performing the greedy rollouts. Moreover, cutting a few hours in this training time is also possible if we do not monitor the evolution of the training on the test sets *(computing the loss on the test sets regularly takes time)*.

For MultiL-MC, we fixed $\mathcal{C}$, the capacity of the replay memory to be equal to 27×256 so that each Monte-Carlo sample is exactly seen 27 times. We used a total of 700 000 episodes here, resulting in an average of 377 000 training steps, which took 56 hours. Indeed, the length of the episodes here is longer on average than the ones used in the curriculum as we directly begin from instances sampled from $\mathbb{D}^{(1)}$ and not the ones where moves were already performed randomly. So the rollout process lasts longer, which is what takes time in our algorithm.

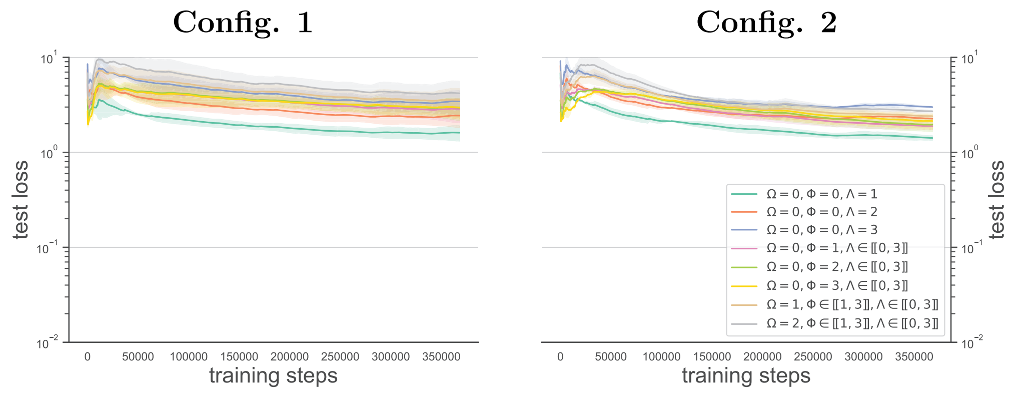
Finally, for MultiL-DQN, we used a capacity $\mathcal{C} = 10\ 240$. In order to perform the same number of training steps for the same number of episodes than the other two algorithms, we generated our data in batches of size of 16: at each time step, there are 16 new instances pushed in memory. We used a total of $16 \times 60\ 000 = 960\ 000$ episodes. The number of training steps performed was 370 000 on average. The time necessary for that was 29 hours. Although this is lower than the other two methods *(due to a much quicker rollout)*, the optimality gap and approximation ratio were so high *(*$\eta = 32.55\%$*,* $\zeta = 1.54$*)* with this amount of data that we actually decided to re-launch an experiment using more episodes. The graphs in Table 4.1 show the behaviour during training of the 3 algorithms with the setting described until now, however the results of optimality gap and approximation ratio for the MultiL-DQN algorithm are those from a different training setting where we used much more episodes. We made a second experiment were we generated batches of size 128 instead of 16, amounting the number of episodes used to 7 680 000 for the same number of training steps. This second experiment took 72 hours, proving that MultiL-DQN actually necessitates way more data than the two other algorithms, for worse results.

For both MultiL-MC and MultiL-DQN, we used a probability ϵ with an exponential decay: $\epsilon_{start} = 0.9$, $\epsilon_{end} = 0.05$ and a temperature $T_{decay} = 1000$.

Discussion. Although the results in Table 4.1 are the outcome of a total of $\sim$ 800000 episodes and $\sim$ 350000 optimization steps for all 3 algorithms, our experiments show that we can divide by 2 the data and 4 the number of steps without sacrificing much the results on $\mathbb{D}^{(1)}$ for the curriculum, which cannot be said of MultiL-DQN that is data hungry, see the next two sub-sections for details. The major drawback of MultiL-Cur is that it needs to compute all possible afterstates at each step of the rollout. This does not scale well with the graph's size: having 100 nodes for the first step means that there are 100 graphs of size 99 to check. Thus, the curriculum we present is a promising step towards automatic design of heuristics, while opening new research directions on restricting the exploration of rollouts.

4.7.2.4. Training the Q network with more data

As discussed earlier, we trained an agent on $\mathbb{D}^{(1)}$ with MultiL-DQN using two configurations. First, we used 960 000 episode for 370 000 optimization steps. Faced with the poor results, we re-trained our agent using more data: 7 680 000 episodes for the same number of steps. We compare the results of the two methods in Table 4.2. We clearly see that training



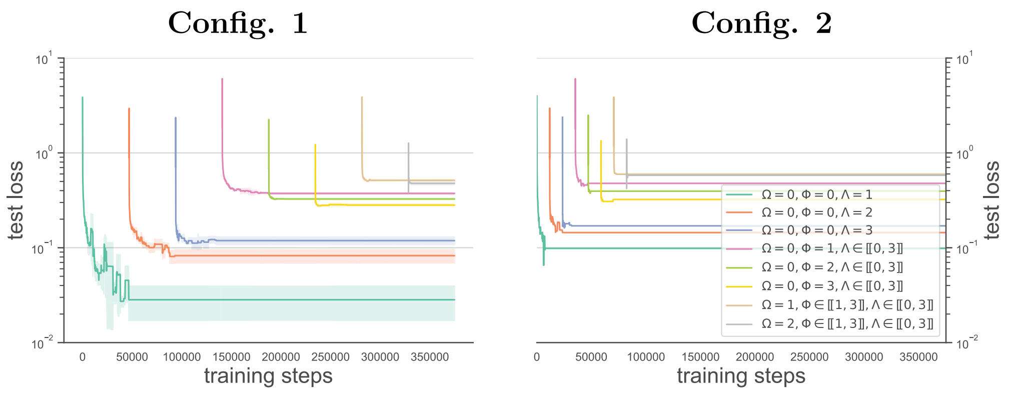
Level	opt gap (%)	approx ratio	opt gap (%)	approx ratio
Vaccination	29.8	1.54	**6.7**	**1.08**
Attack	35.8	1.45	**21.2**	**1.18**
Protection	28.8	1.63	**4.01**	**1.07**

Table 4.2. Comparison between two configurations of training for $\hat{Q}$. In Config. 1, we trained with 960 000 episodes while in Config. 2 we used 7 680 000. We display the evolution of the losses during training on 8 test sets of size 1000. We measure the resulting optimality gap η and approximation ratio ζ on 3 different test sets, one for each of the 3 levels of the problem.

with more data radically impacts the results. More than that, there is a necessity of training with many episodes to obtain reasonable results. We also notice a worse behaviour at the attack stage compared to the other two where it is the defender's turn to play. Thus, we may benefit from adapting the MultiL-DQN algorithms to use two Q networks, one for each player.

4.7.2.5. Training the Value network with less data

In order to assess the capacity of our curriculum to use less data and less training steps, we trained our value network on $\mathbb{D}^{(1)}$ using a second configuration. We re-trained our experts using 50 000 instances in the training sets, with 60 epochs at each stage, instead of 100 000 and 120 originally.



Level	opt gap (%)	approx ratio	opt gap (%)	approx ratio
Vaccination	**0.955**	**1.011**	1.126	1.013
Attack	**0.409**	**1.004**	0.913	1.009
Protection	**0.005**	**1.000**	**0.005**	**1.000**

Table 4.3. Comparison between two configurations of curriculum for $\hat{V}$. In Config. 1, we trained with a total of 800 000 episodes and 375 000 optimization steps while in Config. 2 we used 400 000 episodes and 93 750 steps. We display the evolution of the losses during training on 8 test sets of size 1000 arriving at different stages of the curriculum. We measure the resulting optimality gap η and approximation ratio ζ on 3 different test sets, one for each of the 3 levels of the problem.

The results in Table 4.3 clearly show that training with half the data and a quarter of the steps in the curriculum hardly affects the end results, demonstrating the efficiency of the method. Training with Config. 2 took 15 hours compared to the 36 necessary for Config. 1.

4.7.3. Exploring the abilities of MultiL-Cur

In this section, we provide a more in-depth view of the abilities of MultiL-Cur. In Section 4.7.3.1, we compare the results from the trained agents on $\mathbb{D}^{(2)}$ to 2 other heuristics. Then, in Section 4.7.3.2 we compare the difficulties faced by our curriculum to learn to solve the 3 versions of MCN we considered. In Section 4.7.3.3, we assess the abilities of our trained agents to solve instances out of their training distributions. Finally, in Section 4.7.3.4, we briefly present how one can use the trained value networks to identify multiple solutions to the MCN.

4.7.3.1. Comparison with other heuristics

Results from the last section show that MultiL-Cur is the best performing algorithm on $\mathbb{D}^{(1)}$. Thus, we trained our learning agent with it on $\mathbb{D}^{(2)}$ and tested its performance on the datasets generated in [Baggio et al., 2020]. We compare the results with 2 other heuristics: the random policy *(for each instance, we average the value given by* 10 *random episodes)*, and the DA-AD heuristic [Baggio et al., 2020]. The latter consists in separately solving the two bilevel problems inside MCN: D is chosen by setting Λ to 0 and exactly solving the *Defender-Attacker* problem, while I and P are determined by solving the subsequent *Attacker-Defender* problem. In Table 4.4, we report the inference times t in seconds for our trained agents. The ones for the exact method and DA-AD are from [Baggio et al., 2020].

	MCN									**MCN$_{dir}$**		**MCN$_w$**	
	exact	*random*		*DA-AD*			*MultiL-Cur*			*MultiL-Cur*		*MultiL-Cur*	
$\|V\|$	$t(s)$	$\eta(\%)$	ζ	$t(s)$	$\eta(\%)$	ζ	$t(s)$	$\eta(\%)$	ζ	$\eta(\%)$	ζ	$\eta(\%)$	ζ
20	29	68	3.32	6	**0.3**	**1.00**	**0.4**	0.5	1.00	5.7	1.07	6.9	1.07
40	241	52	2.64	13	7.6	1.09	**0.9**	**5.0**	**1.06**	11.9	1.13	6.5	1.07
60	405	68	3.24	38	7.3	1.09	**1.5**	**4.4**	**1.05**	4.4	1.05	3.7	1.04
80	636	55	2.28	60	3.8	1.04	**2.8**	**2.7**	**1.03**	1.6	1.02	2.8	1.03
100	848	45	1.86	207	**2.7**	**1.03**	**8.7**	49.6	1.50	1.8	1.02	4.1	1.05

Table 4.4. Comparison between several heuristics and exact methods. Results on MCN are computed on the dataset of the original paper [Baggio et al., 2020]. For MCN$_{dir}$ and MCN$_w$, we generated our own datasets using the modification described in Section 4.7.2.2.

The size of the training sets considered in Table 4.4 are describe in Table 4.5.

	size of $\mathcal{D}_{test}$		
$\|V\|$	MCN	MCN$_{dir}$	MCN$_w$
20	120	36	36
40	876	35	34
60	110	23	29
80	101	12	30
100	85	11	27

Table 4.5. Sizes of the test sets used.

Discussion. Table 4.4 reveals that the results given by the MultiL-Cur algorithm are close to the optimum for a fraction of the time necessary to both DA-AD and the quickest exact solver known *(MCN^{MIX}, presented in [Baggio et al., 2020])*. For the MCN instances, the jump in the metrics for graphs of size 100 is due to one outlier among the 85 exactly solved instances of this size. When removed, the values of η and ζ drop to 17.8 and 1.18. The performances measured are consistent accross different problems as we also report low values of η and ζ for MCN$_{dir}$ and MCN$_w$. The curriculum we devised is thus a robust and efficient way to train agents in a Multilevel Budgeted setting.

4.7.3.2. On the difficulty to learn to solve the 3 problems

In this part, we propose to compare the difficulty our curriculum has on learning to solve the 3 different problems MCN, MCN$_{dir}$ and MCN$_w$ on instances from $\mathbb{D}^{(1)}$. For that, we ran our curriculum in exactly the same way 3 times, except for the distribution of graphs from which we sampled our instances: undirected with unit weights for the MCN, directed with unit weights for MCN$_{dir}$ and undirected with integer weights for MCN$_w$. In Figure 4.2, we compare the values of the 3 validation losses during the training, along with the values of the approximation ratio ζ and optimality gap η on 3 test sets of 9000 exactly solved instances from $\mathbb{D}^{(1)}$ in Table 4.6.

Problem	**opt gap** (%)	**approx ratio**
MCN$_w$	7.08	1.069
MCN$_{dir}$	2.84	1.032
MCN	0.51	1.006

Table 4.6. Values of the approximation ratio and optimality gap on a test sets of exactly solved instances from $\mathbb{D}^{(1)}$ for each of the 3 problems.

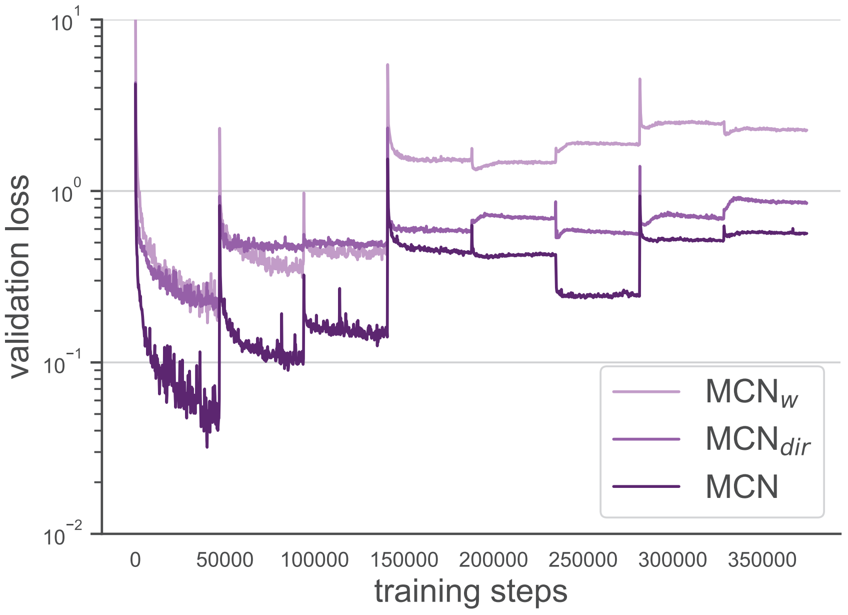


Figure 4.2. Evolution of the loss on the successive validation sets during the curriculum for the 3 problems considered.

Both the table and the figure seem to tell the same story: the easiest problem to learn to solve with our curriculum is the MCN, followed by MCN$_{dir}$, the hardest one being MCN$_w$.

4.7.3.3. Assessing the ability to generalize to larger graphs

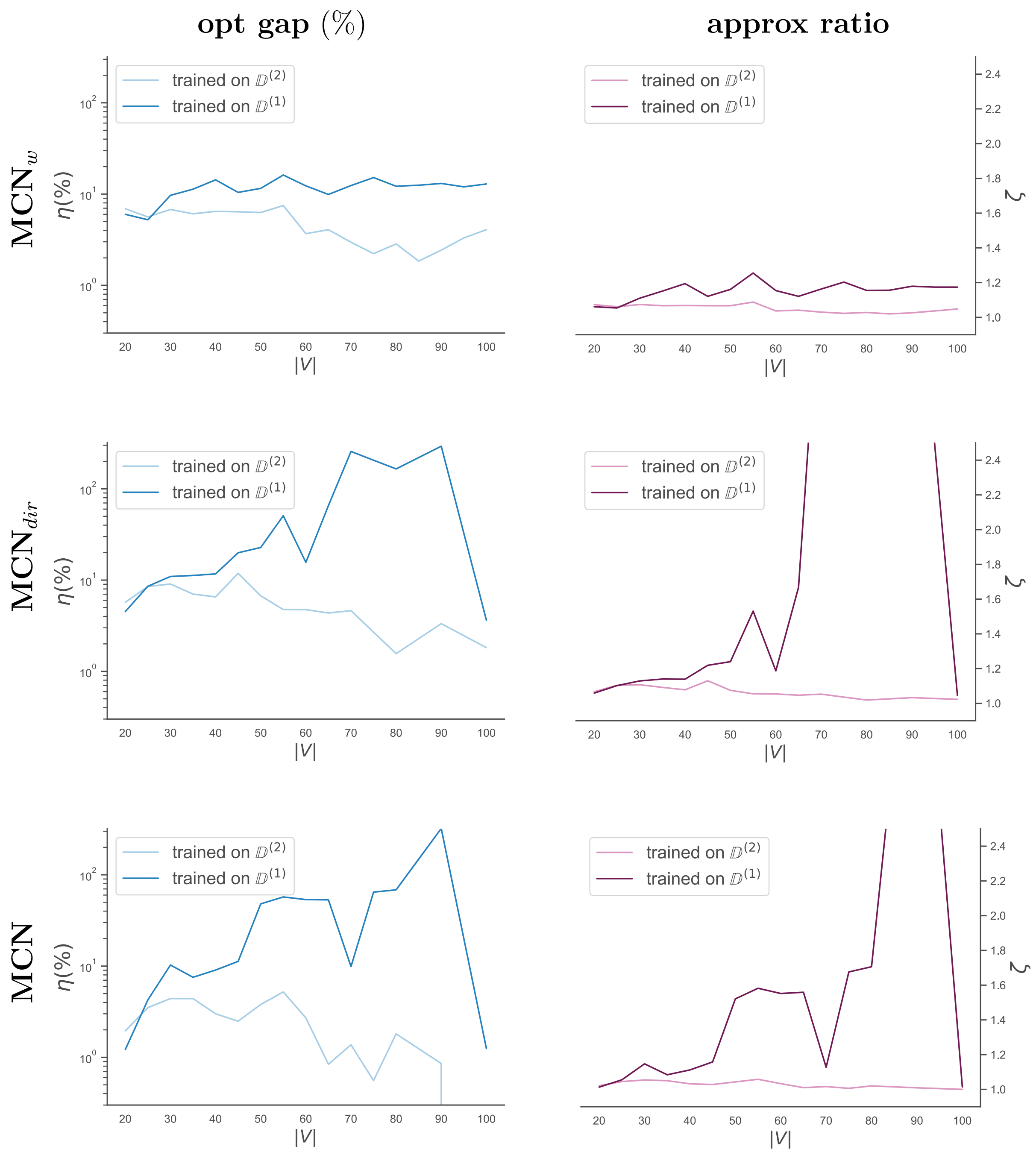


Table 4.7. Evolution of the optimality gap η and the approximation ratio ζ with the size of the graphs at test time for each of the 3 problems considered.

Previous work on learning to solve single level combinatorial problems with graph neural networks such as [Khalil et al., 2017; Kool et al., 2019] reported that their trained agent managed to satisfyingly solve instances with larger graphs at test time than the ones used in their training distributions. In order to assess if this holds for agents trained with our curriculum on the multilevel combinatorial problem, we trained, for each of the 3 problems, our agents on both $\mathbb{D}^{(1)}$ and $\mathbb{D}^{(2)}$, then measured how well they behaved on increasingly larger graphs at test time. We report our results in Table 4.7.

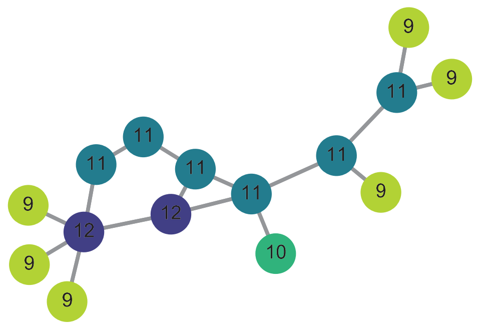
$\lvert V \rvert =$	20	25	30	35	40	45	50	55	60	65	70	75	80	85	90	95	100
MCN$_w$	36	36	36	35	34	33	29	29	29	28	31	29	30	28	29	29	27
MCN$_{dir}$	36	36	36	34	35	32	29	27	23	17	13	-	12	-	10	-	11
MCN	36	36	36	32	30	27	29	26	24	22	17	15	14	-	9	-	10

Table 4.8. Sizes of the test sets used for the results in Table 4.7.

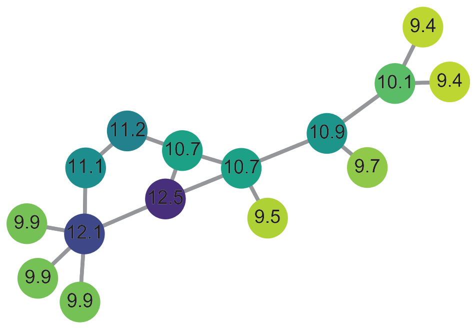
We clearly see in Table 4.7 that the experts trained on $\mathbb{D}^{(2)}$, *i.e.* on larger graphs, perform better than the ones trained on $\mathbb{D}^{(1)}$. From the curves, it seems that our experts can generalize to graphs up to 2 times larger than the ones they were trained on. The fact that for the curves about $\mathbb{D}^{(1)}$ there is first an increase of the values of the metrics and a sudden decrease around $|V| = 80$ may be explained by the fact that η and ζ do not directly measure the goodness of our heuristics. Indeed, if we were to measure how good the decisions taken at a certain level are, we should solve to optimality the subsequent lower levels, which is not what we do here: we use our heuristics everywhere. Thus, when our heuristics perform too badly at *each* level, *i.e.* defending poorly *but also* attacking poorly, there is a chance that the *value* measured in the end of the game is actually not too far from the one that would have followed optimal decisions. To produce the graphs in Table 4.7, we generated 3 test sets, one for each problem, using the solver described in Section 4.7.2.2 with IBM ILOG CPLEX 12.9.0. The number of instances in those datasets for each value of $|V|$ are listed in Table 4.8.

4.7.3.4. Identifying multiple optimal solutions

In many situations, there exists multiple solutions to an instance of a combinatorial problem. For some methods, this can represent a challenge as it clouds the decision-making process [Li et al., 2018]. However, being able to produce multiple optimal solutions to a combinatorial problem is of interest. Here, the formulation we used naturally allows to identify many of the optimal solutions, assuming our value networks correctly approximate the values of each afterstate. Indeed, if our agents correctly label each node with its value *(*i.e. *the value of the afterstate if the action is performed on the node, plus reward)*, then identifying all the possible ways of acting optimaly is directly readable from them, as shown in the example presented in Figure 4.3.



(a) Exact values



(b) Approximate values

Figure 4.3. Exact values and approximate values on an instance of MCN constituted of a graph G and budgets $\Omega = 1$, $\Phi = 1$, $\Lambda = 2$. The exact value of each node is obtained by removing *(vaccinating)* the said node from G and solving exactly the subsequent afterstate with Ω set to 0. The approximate values are obtained by feeding the afterstates to the expert trained on budgets $\Phi = 1$, $\Lambda \in [\![0,3]\!]$ during the curriculum for instances from $\mathbb{D}^{(1)}$.

In Figure 4.3, there are 2 optimal vaccinations: the two blue nodes with value 12. Although the approximate values are not perfectly aligned with the exact ones, the two optimal decisions are clearly identifiable from them, demonstrating the ability of our method to detect multiple optimal solutions.

4.8. Conclusion

The methodology presented in this chapter bridges different research fields: Combinatorial Optimization, Game Theory and Reinforcement Learning. Indeed, it contributes for tackling Combinatorial Optimization problems where players rational behavior is considered, while acknowledging the intractability of exact solutions and, consequently, taking advantage of the recent advances in Reinforcement Learning to approximate them. To do that, we framed the general class of Multilevel Budgeted Combinatorial problems in the Alternating Markov Game framework. This allowed us to derive three algorithms: MultiL-DQN, MultiL-MC and MultiL-Cur. We compared their abilities to solve the MCN and concluded that the curriculum based method gives the best results among the three. We explored some of the properties of MultiL-Cur and showed that, not only does it proposes good quality strategies for a fraction of the time necessary to both exact solvers and previous heuristics for the MCN, but also exhibits interesting behaviour as it allows to identify multiple optimal solutions to a combinatorial problem. Thus, the framework proposed in this chapter is an interesting avenue to explore the automatic design of heuristics for multilevel combinatorial problems.

Conclusion and future research

In this thesis, we conducted an extensive study of a trilevel combinatorial optimization problem on a graph: the Multilvel Critical Node problem (MCN). This problem, introduced in [Baggio et al., 2020], has many connections to recent hot topics in Computer Science as it is linked to Interdiction Games and the detection of critical infrastructures. We highlighted the difficulties encountered by current state-of-the-art methods to solve this problem and proposed to explain them by demonstrating the inherent intractability of the MCN. After introducing notions of computational complexity theory, we proved that several versions of the MCN, as well as some of its sub-problems, are complete for different levels of the Polynomial Hierarchy. Our contributions thus answers the previously open question on the complexity of the MCN, and it contributes to the extension of the list of Σ_2^p-hard problems and the the short list of Σ_3^p-complete problems. However, future directions of research remain opened regarding the complexity of the MCN. Indeed, we still don't know whether or not the unweighted and undirected MCN is Σ_p^2-hard and if the unweighted directed case is Σ_p^3-hard. Moreover, an interesting direction would be to explore the existence of polynomial time approximation schemes for MCN and its sub-problems.

The complexity results motivated our quest to design efficient heuristics for the game. Driven by recent breakthroughs in the field of *Deep Learning for Combinatorial Optimization*, we decided to investigate whether we could design a framework allowing us to *learn* how to solve instances of the MCN. In that respect, we introduced the necessary background in Reinforcement Learning and Graph Neural Networks. But contrary to the combinatorial problems dealt with in previous work, the MCN is not a game involving a single agent. Thus, we devised new methods based on the theory of Alternating Markov Games in the hope to broaden the span of combinatorial problems tackled with Deep Learning by making it reach the shores of the multilevel programming ones. By taking advantages of some intrinsic properties of combinatorial problems, we derived a curriculum allowing a Graph Neural Network based agent to gradually learn to solve problems with an increasing number of levels. Although the framework we devised is set for the general family of Multilevel Budgeted Problems, we only tested it on versions of the MCN. We report promising results compared to other heuristics for the MCN, and shed lights on some of the properties of interest of our curriculum.

However, many challenges remain to be solved in future work. Indeed, the curriculum we devised hardly generalizes to situations with large graphs as our method is based on an *afterstate value network*, making the computation of the values of state-action pairs during rollout burdensome as soon as large instances are tackled. Moreover, further study on the metrics needed to evaluate the quality of the proposed solutions in this multilevel setting is necessary to properly report how close to optimality our agent act at each level. Indeed, the main drawback of heuristics for multilevel optimization is on their evaluation: given a leader's strategy, its associated reward (value) can only be evaluated if the remaining levels are solved to optimality. This means that in opposition to single-level optimization, one must be very careful on the interpretation of the estimated reward associated with an heuristic method: we can be overestimating or underestimating it. In other words, it means that in the remaining levels, players might be failing to behave in an optimal way. Consequently, further research is necessary to provide guarantees on the quality of the obtained solution, namely, on dual bounds.

Overall, this work allowed to extend the relatively small family of complete problems for higher levels than NP in the Polynomial Hierarchy and proposed the first study of its kind in how to tackle multilevel combinatorial problems with Deep Learning by bridging the gap between Multilevel Programming and Alternative Markov Games. It also opened the door for new interesting avenues in this growing field, paving the way for exciting future work.

References

Addis, B., Di Summa, M., and Grosso, A. (2013). Identifying critical nodes in undirected graphs: Complexity results and polynomial algorithms for the case of bounded treewidth. *Discrete Applied Mathematics*, 161(16):2349–2360.

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. *arXiv preprint arXiv:1907.10902*.

Arulselvan, A., Commander, C. W., Elefteriadou, L., and Pardalos, P. M. (2009). Detecting critical nodes in sparse graphs. *Computers & Operations Research*, 36(7):2193 – 2200.

Baggio, A., Carvalho, M., Lodi, A., and Tramontani, A. (2020). Multilevel approaches for the critical node problem. *Operations Research*, To appear.

Bai, Y., Xu, D., Wang, A., Gu, K., Wu, X., Marinovic, A., Ro, C., Sun, Y., and Wang, W. (2020). Fast detection of maximum common subgraph via deep q-learning.

Balcan, M.-F., Dick, T., Sandholm, T., and Vitercik, E. (2018). Learning to branch. In Dy, J. and Krause, A., editors, *Proceedings of the 35th International Conference on Machine Learning*, volume 80 of *Proceedings of Machine Learning Research*, pages 344–353, Stockholmsmässan, Stockholm Sweden. PMLR.

Barrett, T. D., Clements, W. R., Foerster, J. N., and Lvovsky, A. I. (2020). Exploratory combinatorial optimization with reinforcement learning. In *Proceedings of the 34th National Conference on Artificial Intelligence, AAAI*.

Bello, I., Pham, H., Le, Q. V., Norouzi, M., and Bengio, S. (2016). Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*.

Bengio, Y., Lodi, A., and Prouvost, A. (2018). Machine learning for combinatorial optimization: a methodological tour d'horizon.

Bengio, Y., Louradour, J., Collobert, R., and Weston, J. (2009). Curriculum learning. In *Proceedings of the 26th Annual International Conference on Machine Learning*, ICML '09, page 41–48, New York, NY, USA. Association for Computing Machinery.

Blair, C. (1992). The computational complexity of multi-level linear programs. *Annals of Operations Research*, 34(1):13–19.

Bracken, J. and McGill, J. T. (1973). Mathematical programs with optimization problems in the constraints. *Operations Research*, 21(1):37–44.

Brown, G., Carlyle, M., Salmerón, J., and Wood, R. (2006). Defending critical infrastructure. *Interfaces*, 36:530–544.

Bruna, J., Zaremba, W., Szlam, A., and Lecun, Y. (2014). Spectral networks and locally connected networks on graphs. In *International Conference on Learning Representations (ICLR2014), CBLS, April 2014*.

Cai, C. and Wang, Y. (2018). A simple yet effective baseline for non-attributed graph classification. *arXiv preprint arXiv:1811.03508*.

Candler, W. and Norton, R. (1977). Multilevel programming and development policy. Technical Report 258, World Bank Development Research Center.

Cappart, Q., Goutierre, E., Bergman, D., and Rousseau, L.-M. (2019). Improving optimization bounds using machine learning: Decision diagrams meet deep reinforcement learning. In *AAAI*.

Caprara, A., Carvalho, M., Lodi, A., and Woeginger, G. J. (2014). A study on the computational complexity of the bilevel knapsack problem. *SIAM Journal of Optimization*, 24:823–838.

Carvalho, M., Glorie, K., Klimentova, X., Constantino, M., and Viana, A. (2020). Robust models for the kidney exchange problem. *INFORMS Journal on Computing (to appear)*.

Carvalho, M., Lodi, A., and Marcotte, P. (2018). A polynomial algorithm for a continuous bilevel knapsack problem. *Operations Research Letters*, 46(2):185 – 188.

Cook, S. A. (1971). The complexity of theorem-proving procedures. In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, STOC '71, page 151–158, New York, NY, USA. Association for Computing Machinery.

Cormen, T. H., Leiserson, C. E., Rivest, R. L., and Stein, C. (2009). *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition.

Dai, H., Dai, B., and Song, L. (2016). Discriminative embeddings of latent variable models for structured data. In Balcan, M. F. and Weinberger, K. Q., editors, *Proceedings of The 33rd International Conference on Machine Learning*, volume 48 of *Proceedings of Machine Learning Research*, pages 2702–2711, New York, New York, USA. PMLR.

DeNegre, S. (2011). *Interdiction and Discrete Bilevel Linear Programming*. PhD thesis, Lehigh University.

Dudás, T., Klinz, B., and Woeginger, G. J. (1998). The computational complexity of multilevel bottleneck programming problems. In Migdalas, A., Pardalos, P. M., and Värbrand, P., editors, *Multilevel Optimization: Algorithms and Application*, pages 165–179, Boston, MA. Springer US.

Erdos, P. and Renyi, A. (1960). On the evolution of random graphs. *Publ. Math. Inst. Hungary. Acad. Sci.*, 5:17–61.

Etheve, M., Alès, Z., Bissuel, C., Juan, O., and Kedad-Sidhoum, S. (2020). Reinforcement learning for variable selection in a branch and bound algorithm. *arXiv preprint* *arXiv:2005.10026*.

Fan, J., Wang, Z., Xie, Y., and Yang, Z. (2019). A theoretical analysis of deep q-learning. *arXiv preprint arXiv:1901.00137*.

Fey, M. and Lenssen, J. E. (2019). Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*.

Finbow, S., King, A., MacGillivray, G., and Rizzi, R. (2007). The firefighter problem for graphs of maximum degree three. *Discrete Mathematics*, 307(16):2094–2105.

Fischetti, M., Ljubic, I., Monaci, M., and Sinnl, M. (2017). A new general-purpose algorithm for mixed-integer bilevel linear programs. *Operations Research*, 65.

Fischetti, M., Ljubić, I., Monaci, M., and Sinnl, M. (2019). Interdiction games and monotonicity, with application to knapsack problems. *INFORMS Journal on Computing*, 31(2):390–410.

Fischetti, M., Monaci, M., and Sinnl, M. (2018). A dynamic reformulation heuristic for generalized interdiction problems. *European Journal of Operational Research*, 267(1):40 – 51.

Forghani, A., Dehghanian, F., Salari, M., and Ghiami, Y. (2020). A bi-level model and solution methods for partial interdiction problem on capacitated hierarchical facilities. *Computers & Operations Research*, 114:104831.

Furini, F., Ljubic, I., Malaguti, E., and Paronuzzi, P. (2019a). Casting light on the hidden bilevel combinatorial structure of the k-vertex separator problem. In *OR-19-6, DEI, University of Bologna*.

Furini, F., Ljubic, I., Martin, S., and San Segundo, P. (2019b). The maximum clique interdiction problem. *European Journal of Operational Research*, 277:112–127.

Garey, M. R. and Johnson, D. S. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., USA.

Gasse, M., Chételat, D., Ferroni, N., Charlin, L., and Lodi, A. (2019). Exact combinatorial optimization with graph convolutional neural networks. In *NeurIPS*, pages 15554–15566.

Gonzalez, T. F. (2007). *Handbook of Approximation Algorithms and Metaheuristics (Chapman & Hall/Crc Computer & Information Science Series)*. Chapman & Hall/CRC.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. `http://www.deeplearningbook.org`.

Hamilton, W. L. (2020). Lecture notes in graph representation learning, comp 766. `https://cs.mcgill.ca/~wlh/comp766/notes.html`.

Hamilton, W. L., Ying, R., and Leskovec, J. (2017). Inductive representation learning on large graphs. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 1025–1035, Red Hook, NY, USA. Curran Associates Inc.

He, K., Zhang, X., Ren, S., and Sun, J. (2016). Deep residual learning for image recognition. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 770–778.

He, X., Li, C., Huang, T., Li, C., and Huang, J. (2014). A recurrent neural network for solving bilevel linear programming problem. *IEEE Transactions on Neural Networks and Learning Systems*, 25(4):824–830.

Ioffe, S. and Szegedy, C. (2015). Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd International Conference on International Conference on Machine Learning - Volume 37*, ICML'15, page 448–456. JMLR.org.

Jeroslow, R. G. (1985). The polynomial hierarchy and a simple model for competitive analysis. *Mathematical Programming*, 32(2):146–164.

Johannes, B. (2011). *New Classes of Complete Problems for the Second Level of the Polynomial Hierarchy.* PhD thesis, Technischen Universitat Berlin.

Johnson, D. S. (2012). A brief history of np-completeness, 1954–2012. *Optimization Stories, Special Volume of Documenta Mathematica*, pages 359–376.

Karp, R. M. (1972). *Reducibility among Combinatorial Problems*, pages 85–103. Springer US, Boston, MA.

Khalil, E., Dai, H., Zhang, Y., Dilkina, B., and Song, L. (2017). Learning combinatorial optimization algorithms over graphs. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 6348–6358. Curran Associates, Inc.

Kingma, D. P. and Ba, J. (2015). Adam: A method for stochastic optimization. In Bengio, Y. and LeCun, Y., editors, *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings.*

Klicpera, J., Bojchevski, A., and Günnemann, S. (2019). Combining neural networks with personalized pagerank for classification on graphs. In *International Conference on Learning Representations.*

Kool, W., van Hoof, H., and Welling, M. (2019). Attention, learn to solve routing problems! In *International Conference on Learning Representations.*

Lachhwani, K. and Dwivedi, A. (2017). Bi-level and multi-level programming problems: Taxonomy of literature review and research issues. *Archives of Computational Methods in Engineering*, 25.

Lalou, M., Tahraoui, M. A., and Kheddouci, H. (2018). The critical node detection problem in networks: A survey. *Computer Science Review*, 28:92 – 117.

Lange, M. D., Aljundi, R., Masana, M., Parisot, S., Jia, X., Leonardis, A., Slabaugh, G., and Tuytelaars, T. (2019). A continual learning survey: Defying forgetting in classification tasks. *arXiv preprint arXiv:1909.08383.*

Lee, J., Lee, I., and Kang, J. (2019). Self-attention graph pooling. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 3734–3743, Long Beach, California, USA. PMLR.

Levin, L. A. (1973). Universal sequential search problems. *Problemy peredachi informatsii*, 9(3):115–116.

Lewis, J. M. and Yannakakis, M. (1980). The node-deletion problem for hereditary properties is np-complete. *Journal of Computer and System Sciences*, 20(2):219–230.

Li, Y., Tarlow, D., Brockschmidt, M., and Zemel, R. (2016). Gated graph sequence neural networks. In *International Conference on Learning Representations*.

Li, Z., Chen, Q., and Koltun, V. (2018). Combinatorial optimization with graph convolutional networks and guided tree search. In Bengio, S., Wallach, H., Larochelle, H., Grauman, K., Cesa-Bianchi, N., and Garnett, R., editors, *Advances in Neural Information Processing Systems 31*, pages 539–548. Curran Associates, Inc.

Littman, M. L. (1994). Markov games as a framework for multi-agent reinforcement learning. In *Proceedings of the Eleventh International Conference on International Conference on Machine Learning*, ICML'94, page 157–163, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

Littman, M. L. (1996). *Algorithms for Sequential Decision-Making*. PhD thesis, Brown University, USA.

Littman, M. L. and Szepesvari, C. (1996). A generalized reinforcement-learning model: Convergence and applications. Technical report, Brown University, USA.

Lodi, A. and Zarpellon, G. (2017). On learning and branching: a survey. *TOP*, 25:207–236.

Lozano, L. and Smith, J. C. (2017). A backward sampling framework for interdiction problems with fortification. *INFORMS J. Comput.*, 29:123–139.

Ma, Q., Ge, S., He, D., Thaker, D., and Drori, I. (2019). Combinatorial optimization by graph pointer networks and hierarchical reinforcement learning. *arXiv preprint arXiv:1911.04936*.

Martin, P. (2007). Tri-level optimization models to defend critical infrastructure. Master's thesis, Naval Postgraduate School, Monterey.

Mazyavkina, N., Sviridov, S., Ivanov, S., and Burnaev, E. (2020). Reinforcement learning for combinatorial optimization: A survey.

McCormick, G. P. (1976). Computability of global solutions to factorable nonconvex programs: Part i — convex underestimating problems. *Mathematical Programming*, 10(1):147–175.

Meyer, A. R. and Stockmeyer, L. J. (1972). The equivalence problem for regular expressions with squaring requires exponential space. In *13th Annual Symposium on Switching and Automata Theory (swat 1972)*, pages 125–129.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., Graves, A., Riedmiller, M., Fidjeland, A. K., Ostrovski, G., Petersen, S., Beattie, C., Sadik, A., Antonoglou, I., King, H., Kumaran, D., Wierstra, D., Legg, S., and Hassabis, D. (2015). Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533.

Nabli, A. and Carvalho, M. (2020). Curriculum learning for multilevel budgeted combinatorial problems. *arXiv preprint arXiv:2007.03151*.

Nabli, A., Carvalho, M., and Hosteins, P. (2020). Complexity of the multilevel critical node problem. *arXiv preprint arXiv:2007.02370*.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Puterman, M. L. (1994). *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., USA, 1st edition.

Ratliff, H. D., Sicilia, G. T., and Lubore, S. H. (1975). Finding the n most vital links in flow networks. *Management Science*, 21(5):531–539.

Schaefer, M. and Umans, C. (2002). Completeness in the polynomial-time hierarchy a compendium. *Sigact News - SIGACT*, 33.

Shapley, L. S. (1953). Stochastic games. *Proceedings of the National Academy of Sciences*, 39(10):1095–1100.

Shen, S., Smith, J. C., and Goli, R. (2012). Exact interdiction models and algorithms for disconnecting networks via node deletions. *Discrete Optimization*, 9(3):172–188.

Shih, H.-S., Wen, U.-P., Lee, S., Lan, K.-M., and Hsiao, H.-C. (2004). A neural network approach to multiobjective and multilevel programming problems. *Comput. Math. Appl.*, 48(1–2):95–108.

Shoham, Y., Powers, R., and Grenager, T. (2007). If multi-agent learning is the answer, what is the question? *Artificial Intelligence*, 171(7):365 – 377. Foundations of Multi-Agent Learning.

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., Hubert, T., Baker, L., Lai, M., Bolton, A., Chen, Y., Lillicrap, T., Hui, F., Sifre, L., van den Driessche, G., Graepel, T., and Hassabis, D. (2017). Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359.

Sinha, A., Malo, P., and Deb, K. (2018). A review on bilevel optimization: From classical to evolutionary approaches and applications. *IEEE Transactions on Evolutionary Computation*, 22(2):276–295.

Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.*, 15(1):1929–1958.

Stockmeyer, L. J. and Meyer, A. R. (1973). Word problems requiring exponential time(preliminary report). In *Proceedings of the Fifth Annual ACM Symposium on Theory of Computing*, STOC '73, page 1–9, New York, NY, USA. Association for Computing Machinery.

Sutton, R. S. and Barto, A. G. (1998). *Introduction to Reinforcement Learning.* MIT Press, Cambridge, MA, USA, 1st edition.

Tahernejad, S., Ralphs, T. K., and DeNegre, S. T. (2020). A branch-and-cut algorithm for mixed integer bilevel linear optimization problems and its implementation.

Talbi, E.-G. (2013). *Metaheuristics for Bi-level Optimization.* Springer-Verlag Berlin Heidelberg.

Tesauro, G. (2002). Programming backgammon using self-teaching neural nets. *Artificial Intelligence*, 134(1):181 – 199.

Turing, A. M. (1936). On Computable Numbers, with an Application to the Entscheidungsproblem. *Proceedings of the London Mathematical Society*, s2-42(1):230–265.

Velicković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. (2018). Graph attention networks. In *International Conference on Learning Representations.*

Ventresca, M., Harrison, K. R., and Ombuki-Berman, B. M. (2018). The bi-objective critical node detection problem. *European Journal of Operational Research*, 265(3):895–908.

von Stackelberg, H. (1934). *Marktform und Gleichgewicht.* Springer-Verlag, Berlin.

Watkins, C. J. C. H. (1989). *Learning from Delayed Rewards.* PhD thesis, King's College, Cambridge, UK.

Wood, R. (1993). Deterministic network interdiction. *Mathematical and Computer Modelling*, 17(2):1–18.

Wrathall, C. (1976). Complete sets and the polynomial-time hierarchy. *Theoretical Computer Science*, 3(1):23–33.

Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., and Yu, P. S. (2020). A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, page 1–21.

Yannakakis, M. (1978). Node-and edge-deletion np-complete problems. In *Proceedings of the Tenth Annual ACM Symposium on Theory of Computing*, STOC '78, page 253–264. Association for Computing Machinery.

Zaheer, M., Kottur, S., Ravanbakhsh, S., Poczos, B., Salakhutdinov, R. R., and Smola, A. J. (2017). Deep sets. In Guyon, I., Luxburg, U. V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., and Garnett, R., editors, *Advances in Neural Information Processing Systems 30*, pages 3391–3401. Curran Associates, Inc.

Zarpellon, G., Jo, J., Lodi, A., and Bengio, Y. (2020). Parameterizing branch-and-bound search trees to learn branching policies. *arXiv preprint arXiv:2002.05120*.

Zhang, G., Lu, J., and Gao, Y. (2015). *Multi-Level Decision Making: Models, Methods and Applications*. Springer-Verlag Berlin Heidelberg.