UNIVERSITÀ
di VERONA

Combining Optimization and Machine Learning for the Formation of Collectives

Adrià Fenoy Barceló

Ph.D. Thesis

Advisors:

Alessandro Farinelli

Filippo Bistaffa

Combining Optimization and Machine Learning
for the Formation of Collectives

Adrià Fenoy Barceló

Ph.D. thesis

Verona, 7 March 2023

Abstract

This thesis considers the problem of forming collectives of agents for real-world applications aligned with Sustainable Development Goals (e.g., shared mobility and cooperative learning). Such problems require fast approaches that can produce solutions of high quality for hundreds of agents. With this goal in mind, existing solutions for the formation of collectives focus on enhancing the optimization approach by exploiting the characteristics of a domain. However, the resulting approaches rely on specific domain knowledge and are not transferable to other collective formation problems. Therefore, approaches that can be applied to various problems need to be studied in order to obtain general approaches that do not require prior knowledge of the domain.

Along these lines, this thesis proposes a general approach for the formation of collectives based on a novel combination of machine learning and an *Integer Linear Program*. More precisely, a machine learning component is trained to generate a set of promising collectives that are likely to be part of a solution. Then, such collectives and their corresponding utility values are introduced into an *Integer Linear Program* which finds a solution to the collective formation problem. In that way, the machine learning component learns the structure shared by "good" collectives in a particular domain, making the whole approach valid for various applications.

In addition, the empirical analysis conducted on two real-world domains (i.e., ridesharing and team formation) shows that the proposed approach pro-

vides solutions of comparable quality to state-of-the-art approaches specific to each domain.

Finally, this thesis also shows that the proposed approach can be extended to problems that combine the formation of collectives with other optimization objectives. Thus, this thesis proposes an extension of the collective formation approach for assigning pickup and delivery locations to robots in a warehouse environment. The experimental evaluation shows that, although it is possible to use the collective formation approach for that purpose, several improvements are required to compete with state-of-the-art approaches.

Overall, this thesis aims to demonstrate that machine learning can be successfully intertwined with classical optimization approaches for the formation of collectives by learning the structure of a domain, reducing the need for ad-hoc algorithms devised for a specific application.

Contents

Chapter 1

Introduction

Choosing a line to stand in the supermarket, organizing activities during vacation trips, and thinking about the best way to get to work are examples of everyday decisions that involve choosing, among several alternatives, the one facilitating daily life. In the academic literature, problems involving finding a solution that stands over others are known as optimization problems. In addition, if the solutions are obtained by considering a combination of different elements, the problem is called combinatorial, e.g. choosing a set of cities to visit during a road trip requires considering different possible combinations of cities. Over the last century, the industry has used novel techniques emerging from academic research to solve combinatorial optimization problems in order to automate tasks such as resource allocation, scheduling, and routing, among others. In consequence, the impact of research in combinatorial optimization has promoted not only sustainable economic growth but also an improvement in the quality of life and work environments. Actually, research in this area has been aligned with UN sustainability goals to bring solutions to current societal problems, such as building sustainable communities, promoting sustainable production and consumption patterns, and ensuring access to affordable and sustainable sources of energy and water.

Combinatorial optimization is one of the most active research topics in artificial intelligence because of how challenging and relevant these problems are. On the one hand, many real-world combinatorial optimization scenarios involve hundreds of elements whose combinations represent the space of possible solutions. Because the number of combinations grows exponentially with the *scale* of the problem, i.e., the number of elements to be combined, finding a solution by examining all possible solutions would take exponentially increasing time as the scale of the problem grows. Although scalability issues can be partially addressed by algorithms adopting dynamic programming strategies, these algorithms aiming for optimal solutions cannot be applied to large-scale scenarios with hundreds of elements. Moreover, because combinatorial optimization problems are known to be *NP-hard* (Papadimitriou and Steiglitz, 1998), algorithms computing optimal solutions in polynomial time are not known to exist. Therefore, research is constantly developing novel algorithmic solutions that study approximate solution approaches for these large-scale scenarios. On the other hand, combinatorial optimization problems frequently include requirements in the form of constraints that its solution must fulfil. For example, when organizing activities during vacation trips, these need to be scheduled considering weather conditions if they are held outdoors. One strand of literature focuses on efficiently incorporating constraints into algorithmic solutions.

In many relevant scenarios, the elements that must be combined to find a solution are multiple computational units known as agents. In such *multi-agent systems*, different agents pursuing a common objective need to establish behavioural rules that guide them in order to meet their goals (Cerquides et al., 2014). Developing methods to provide such agents with a way to behave collectively sets up the combinatorial optimization problem studied in this thesis, i.e., *Collective Formation*. As a combinatorial optimization problem,

collective formation faces the same scalability issues and also requires efficient modelling of the problem constraints. As a result, state-of-the-art approaches to collective formation address the scalability issues by solving the problem approximately, that is, by finding solutions close to the optimal one. More precisely, these approaches shorten computation time by settling for suboptimal solutions of acceptable quality, reducing years of computation to seconds for large-scale problems with hundreds of agents. In addition, these approaches often exploit the structure of a particular domain by incorporating ad-hoc instructions that further accelerate the computation. For instance, Bistaffa et al. (2019) propose an approximate solution approach to *ridesharing*, the problem of arranging passengers in cars to reduce their total travel distance and the derived environmental benefits. The authors noted that two car arrangements differing only by one additional passenger would be of similar quality, especially if the new passenger trip can be effectively integrated into the current route. Thus, they propose an incremental approach to form collectives that selects the best passengers to add by means of a greedy approach with a probabilistic component. Andrejczuk et al. (2019) investigate the problem of forming teams of students in a classroom to cover a set of competencies necessary to conduct various tasks. Because teams need to cover all required competencies, they should include complementary student profiles. For this reason, the authors propose a solution that continuously combines and rearranges teams to adapt to the structure of the domain.

A crucial factor for the success of ad-hoc approaches is the ability to exploit domain-specific traits. However, this limits the applicability of the resulting approaches to other collective formation problems with a different structure. Moreover, as a result of the rapid advancement of technological tools and to satisfy the needs of an increasingly interconnected society, new collective formation problems are emerging on a continual basis. These emerging problems

typically lack a previously studied structure that can be immediately exploited. However, they still require fast approaches to produce high-quality solutions for large-scale settings. In this paradigm, general approaches to collective formation are crucial because they provide fast and high-quality solutions without making domain-specific assumptions.

In this regard, machine learning has been successfully employed to build general approaches in different domains such as natural language processing and computer vision. One notorious application of machine learning is in building general large language models (Brown et al., 2020) that exhibit anthropomorphic capabilities: able to handle conversations, generate articles or poetry, follow instructions and even solve simple problems. However, these approaches fail when they are presented with more complex problems as they are not designed to learn decisions solving combinatorial optimization problems successfully.

Nonetheless, as noted by Bengio, Lodi, and Prouvost (2021), machine learning is starting to be employed to learn fast approximations replacing heavy computations and enhancing the optimization approach by learning the structure of a specific domain. For example, Vinyals, Fortunato, and Jaitly (2015) propose *Pointer Networks*, a machine learning approach exhibiting state-of-the-art performance in generating convex hulls, Delaunay triangulations and solving the *travelling salesperson problem*. Kool, Van Hoof, and Welling (2018) propose an approach based on an attention operation (Bahdanau, Cho, and Bengio, 2014) which also exhibits state-of-the-art performance on the *travelling salesperson problem* and the *vehicle routing problem*. While both of these approaches solve the problem in an end-to-end fashion, other ones consider machine learning as a component enhancing the performance of another classical solution approach. For instance, Ding et al. (2020) incorporate *graph convolutional neural networks* to predict the value of decision variables in *mixed-integer*

linear programs. Although some attempts have been made to incorporate machine learning into combinatorial optimization, the use of machine learning for collective formation did not receive significant attention in the literature.

1.1 Contributions

In the applications mentioned above, machine learning has been employed to improve the performance of existing approaches in terms of solution quality and computation time. In addition, in this thesis, machine learning is viewed as a component able to learn the domain-specific structure of diverse collective formation problems to enhance the generalization capability of the optimization approach. To this end, this thesis proposed to train a machine learning component to generate "good" candidate collectives. The candidates are then incorporated into an *integer linear program* (*ILP*) formulation of the collective formation problem, which is solved using conventional techniques. A high-level representation of the proposed approach is depicted in figure 1.1.

Figure 1.1: High-level schema of the proposed approach combining machine learning and an *ILP* for collective formation.

This thesis studies different machine learning alternatives to generate candidate collectives for the proposed approach. In particular, this thesis considers: (i) *generative adversarial networks* (Goodfellow et al., 2014), which are a generative model with a lot of success in image generation; (ii) *pointer networks* a sequence-to-sequence model with some applications in combinatorial optimization; and (iii) an *attention-based* model, which is inspired by the model proposed by Kool, Van Hoof, and Welling (2018) for routing problems.

Different learning methodologies are also discussed; these are: learning from examples (*supervised learning*) and learning from experience (*reinforcement learning*). While a supervised approach can be practical for some applications, it requires examples to train the models. Assuming that a large set of examples can be obtained for any collective formation problem is a step back in terms of the generality of the whole approach. Therefore, reinforcement learning, which does not require external components, is more practical.

In addition, this thesis observes that generating "good" candidate collectives is not enough to produce high-quality collective formation solutions because the candidates produced by the machine learning component tend to be similar. For this reason, this thesis proposes to maximize the entropy of the generated candidates to improve their diversity and enhance the quality of solutions of the general approach.

Two collective formation domains and their respective ad-hoc approaches are considered for testing the performance of the proposed general approach. The first is the ridesharing domain described in Bistaffa et al. (2019), whose *probabilistic-greedy* approach exploits incremental solutions. The second is the team formation domain studied in Andrejczuk et al. (2019), whose *synergistic* approach exploits solutions obtained from recombining other ones. The general approach proposed in this thesis produces solutions of comparable quality to the ones obtained with the ad-hoc approaches. Thus, it succeeds in learning the features associated with high-quality solutions in two structurally different domains. These favourable results towards a general collective formation approach were presented in the *PRL* workshop at *IJCAI 2022*. Moreover, an extended version of the work, including additional experiments on the impact of entropy has been submitted to the *Artificial Intelligence Journal*, receiving a major revision with three positive reviews. A new version addressing the comments provided by the reviewers is under preparation.

With the goal of studying the limitations of the machine learning component in generating collectives for more convoluted domains, an *attention-based* model is devised for the *capacitated multi-agent pickup and delivery* (capacitated *MAPD*) problem. This problem combines the assignment of pickup and delivery tasks to agents with the path-planning of agents. Under this complex scenario, the *attention-based* model is employed to produce collectives of tasks that are continuously assigned to agents, which then plan their path to execute the tasks using a conventional path-planning solver. For the task assignment, which is treated as a collective formation problem, the *ILP* component cannot be employed since the collective values are not known until the path-planning has been executed. Instead, the *attention-based* model is trained to learn the collectives with the best values so as to generate the assignments directly.

1.2 Structure of the Thesis

The rest of this thesis is organized as follows. Chapter 2 presents combinatorial optimization and three combinatorial problems relevant to this work: *Mixed Integer Linear Programs*, the *Travelling Salesperson Problem* and the *Vehicle Routing Problem*. Following, it defines collective formation and classifies existing approaches according to their ability to find optimal solutions into complete, anytime and heuristic approaches. The last part of this chapter explains the meaning of learning algorithmic decisions, learning to generalize and learning constraints. The chapter concludes with a review of the existing machine learning approaches for combinatorial optimization.

Chapter 3 presents the proposed general approach combining machine learning with an *ILP* to solve collective formation problems. This chapter also introduces the different machine learning alternatives for generating collectives (*GANs*, *pointer networks* and *attention-based* models) and the diverse training

methodologies. The experimental evaluation conducted to assess the performance of the proposed approach, along with all its variants, is presented at the end of the chapter, including an extensive analysis to determine the impact of the entropy term.

Chapter 4 introduces an extension of the *attention-based* model to conduct the assignment of tasks inside the capacitated *MAPD* problem. The chapter includes an experimental evaluation comparing the performance of the proposed approach to other assignment algorithms. A discussion on the current limitations of the model and possible ways to extend its applicability to capacitated *MAPD* closes the chapter.

Finally, chapter 5 discusses the contributions of the thesis and revises the key outcomes and limitations of the studied approaches. Moreover, it examines the potential impact of these contributions on future work, laying the groundwork for future research directions aiming at extending this work.

Chapter 2

Background

This chapter contextualizes the goal of this thesis with a formal description of the problems it attempts to solve and discusses the main related approaches, as well as their contributions and limitations to their respective fields. In Section 2.1, it is discussed the combinatorial optimization framework. Section 2.2 delves into the main combinatorial optimization problem tackled by this work, *Collective Formation*, and surveys the most relevant and standard approaches to this problem. Finally, Section 2.4 discusses the application of *Machine Learning* in the combinatorial optimization field, together with different solutions approaches.

2.1 Combinatorial Optimization

Combinatorial optimization problems, which consider tasks such as resource allocation, scheduling, and routing, are among the most common problems encountered in industry and research. These problems aim to find the best arrangement out of several possibilities characterized by discrete variables. In general, a combinatorial optimization problem can be formulated as a mini-

mization problem with constraints (Peres and Castelli, 2021):

$$P = (S, f, \Omega), \tag{2.1}$$

where:

- P is the optimization problem,
- S is the search space of the domain,
- f is the objective function and
- Ω is the set of problem constraints.

The search space S is defined over a set of decision variables $\{X_1, X_2, \ldots\}$, representing the decisions to be made to solve the problem. The set of problem constraints Ω defines a sub-region of the search space where the constraints are fulfilled named the feasible space of solutions $\mathcal{F}_\Omega(S)$. The objective function specifies a metric for assessing the solution quality, i.e., it defines a mapping $f : S \to \mathbb{R}$. In order to solve a combinatorial optimization problem, one has to find a solution in the feasible region $\mathcal{F}_\Omega(S)$ with the minimum objective function:

$$s^* \in \{s \mid f(s) \leq f(s') \ \ \forall s, s' \in \mathcal{F}_\Omega(S)\}. \tag{2.2}$$

Because of the discrete nature of combinatorial optimization problems, it is not possible to use gradient methods (Ruder, 2016), which exploit continuous domains by modifying a solution s in the direction of the gradient, which provides the "fastest improvement" of the objective function. Moreover, the number of decision variables enlarges the search space according to the possible number of combinations between those. Therefore, optimally solving a combinatorial optimization problem becomes combinatorially harder as the problem

size increases. Due to this complexity, most combinatorial optimization problems are classified as NP-hard (Papadimitriou and Steiglitz, 1998), i.e. they do not admit an algorithm which can solve them in polynomial time. In this regard, algorithms for solving combinatorial optimization problems are classified as *complete approaches*, which aim to find the optimal solution, and *approximate approaches*, which settle for "good in practice" solutions in exchange for faster computation time.

Approximate approaches, also known as heuristic approaches, use domain-specific instructions exploiting the inherent structure of the problem to reduce the search space. Some heuristic approaches even provide optimality guarantees (Sandholm et al., 1999; Dang and Jennings, 2004; Rahwan, Michalak, and Jennings, 2011; Rahwan et al., 2009), although they are usually less specialized in the domain and thus slower than those that do not. Occasionally, problem structure can barely be exploited; therefore, designing algorithmic instructions to tackle these problems becomes complicated. In those cases, one might consider using generic strategies for exploring the search space, known as *meta-heuristics*. The advantage of using these approaches is that they take relatively few assumptions about the optimization domain. However, they may also use domain-specific knowledge in the form of heuristics that are controlled by an upper-level strategy (Blum and Roli, 2003). For example, in genetic algorithms, genetic operators controlling the evolution of solutions are usually designed for a particular domain structure (Holland, 1984). In recent years, the interest in machine learning techniques playing the role of heuristics for combinatorial optimization problems has risen in popularity. Section 2.4 presents various approaches following this trend.

The following sections present the specific combinatorial optimization problems constituting the background of this thesis. A *mixed integer linear program* is a widely used approach for diverse problems, and it is also the core of the

solution approach presented in this thesis. The *travelling salesperson problem* and the *vehicle routing problem* are among the most recurrent combinatorial problems tackled with machine learning, and their solution approaches inspire part of the solution provided in this thesis. Finally, *collective formation* is the combinatorial optimization problem for which this thesis aims at obtaining a general solution.

2.1.1 Mixed Integer Linear Programs

An *integer linear program* (*ILP*) solves a combinatorial optimization problem considering a linear optimization function f, linear constraints Ω and integer variables. In addition, if some variables are allowed to be non-integer, the program is known as a *mixed integer linear program* (*MILP*). Formally, a *MILP* in its canonical form is defined as:

$$\begin{aligned} \text{minimize} \quad & \sum_i c_i \cdot x_i, \\ \text{subject to} \quad & \sum_i a_{ij} \cdot x_i \leq b_j \quad \forall j, \end{aligned} \tag{2.3}$$

where $x_i \geq 0$ and $x_i \in \mathbb{Z}$ for all variables x_i.

When all variables are non-integer, the problem can be solved in polynomial time. Otherwise, *MILPs* are known to be NP-hard, but even so, efficient algorithms exist to solve them (Lodi, 2010). These algorithms are based on a hybrid of the *branch-and-bound* and *cutting planes* techniques.

The *branch-and-bound* algorithm (Land and Doig, 2010) starts by considering a relaxation of the *MILP* in which the integrality constraint has been dropped. The resulting problem contains the feasible region of the original problem, and since all of the variables are non-integer, it can be solved in polynomial time. From there, the algorithm iteratively constructs a tree of sub-problems where the root is the relaxation of the original *MILP*. At each

node, the corresponding problem is solved, and two sub-problems are obtained by selecting a variable with a non-integer solution x_i^*, constraining it to be an integer and introducing the following constraints

$$x_i \leq \lfloor x_i^* \rfloor \quad \text{and} \quad x_i \geq \lfloor x_i^* \rfloor + 1. \tag{2.4}$$

A child node is created for each sub-problem which, by construction, has disjoint feasible regions and excludes the solution of the previous problem relaxation. If a sub-problem does not admit a feasible solution or the solution is *MILP-compatible*, the corresponding node is no further expanded. Eventually, when the tree has been fully constructed, the best *MILP-compatible* solution among all the sub-problems is also the solution to the original problem.

The *cutting planes* algorithm (Gomory, 2010) also considers a non-integer relaxation to the problem. In that case, given a solution x^* which is not *MILP-compatible*, the algorithm proposes a "cutting plane" to the search space,

$$\alpha^T x \geq \alpha_0, \tag{2.5}$$

which excludes the infeasible solution, i.e. $\alpha^T x^* < \alpha_0$, but is satisfied by all other feasible solutions. By following this approach, the algorithm builds a convex hull out of *MILP-compatible* solutions, from which it obtains the solution to the original problem.

In practice, both algorithmic strategies can be integrated to reduce the number of nodes that need to be explored in the *branch-and-bound* algorithm by reducing the solution space with *cutting planes*. The resulting approach, known as *branch-and-cut*, is the current state-of-the-art. Henceforth, it is adopted by many commercial and non-commercial *MILP* solvers, which further improve its performance through additional algorithmic components, preprocessing and primal heuristics.

This thesis proposes to formulate the generation of collectives as an *ILP*. Section 2.2.2 explains the problems this formulation presents for collective formation, and chapter 3 provides a solution approach to overcome these limitations by employing machine learning to reduce the search space.

2.1.2 Travelling Salesperson Problem

Given a list of cities P, the *travelling salesperson problem* (*TSP*) aims at finding the shortest possible route that enters and exits each city exactly once. Despite the naive definition of the problem, it appears in many different areas, such as microchip design (Chan and Mercier, 1989), DNA sequencing (Pevzner, Tang, and Waterman, 2001) and astronomy (Bailey, McLain, and Beard, 2001). The problem is known to be NP-hard; in fact, the complexity of the problem arises from the combinatorial number of possible solutions that need to be considered, e.g. $15! = 1.3$ trillion routes for only 15 cities.

The *TSP*, as many other combinatorial optimization problems, admits an *ILP* formulation:

$$\begin{aligned} \text{minimize} \quad & \sum_{i,j} d_{ij} \cdot x_{ij}, \\ \text{subject to} \quad & \sum_{i} x_{ij} = 1 \quad \forall j \in P, \\ & \sum_{j} x_{ij} = 1 \quad \forall i \in P, \\ & \sum_{i,j \in Q} x_{ij} \leq |Q| - 1 \quad \forall Q \subset P, \end{aligned} \tag{2.6}$$

where $i, j \in \mathbb{N}$ are indices referring to each city, d_{ij} indicates the distance from city i to city j, and, conforming a route, $x_{ij} = 1$ if a trip from city i to city j is contained in the route, and $x_{ij} = 0$ otherwise. Regarding the constraints, the first two impose that the route should enter and exit each city exactly once.

The last constraint ensures that no sub-group of cities Q forms a sub-tour, resulting in a solution with disjoint tours.

Most of the complete approaches to the TSP rely on the ILP formulation, which is solved by employing problem-specific *branch-and-cut* algorithms (Lenstra and Shmoys, 2009). While complete approaches can handle instances up to 10^4 cities, heuristic approaches scale up to 10^6 cities while providing solutions that are only $2-3\%$ off the optimal. The most popular heuristic approach for the TSP is the one by Christofides (1976) known as the *Christofides algorithm*, which solution is guaranteed to be better than 1.5 times the optimal one. Moreover, the TSP has motivated the design of some meta-heuristics such as the popular *ant colony optimization* algorithm (Dorigo and Gambardella, 1997). This approach mimics the behaviour of real ants following a trail of pheromones to find short paths between food sources and their nest. Current solutions adopting the ant-colony optimization approach exhibit state-of-the-art performance up to 2.33% off the optimal (Wang and Han, 2021).

2.1.3 Vehicle Routing Problem

The *vehicle routing problem* (*VRP*) is a generalization of the TSP in which a set of vehicles aim at delivering to a set of customers. All the routes start and end at a common depot, and the goal is to reduce the sum of distances travelled by each vehicle. After being defined by Dantzig and Ramser (1959), the *VRP* has been gradually adopted by the transportation industry as a solution to a wide range of applications for which variations to the original problem have been proposed. Examples are the capacitated *VRP* (which includes vehicle capacity constraints), the *VRP* with time windows or the *VRP* with pickup and delivery.

Similar to the TSP, complete approaches for the *VRP* are not practical for large-scale problems. For this reason, heuristic and meta-heuristic approaches

are the ones preferred by commercial solvers. On the one hand, heuristic approaches can achieve high-quality solutions within a modest computation time. Furthermore, most of them are easily extensible to account for the wide range of real-world constraints. On the other hand, meta-heuristics place a greater emphasis on deep exploration of interesting regions in the solution space, resulting in longer computation times but higher quality solutions.

2.2 Collective Formation

Many problems consider the existence of different agents with a common goal. Depending on the application domain, agents may pursue altruistic or selfish goals. For example, in ridesharing (Bistaffa et al., 2019), agents aim to achieve an environmental benefit in favour of shorter travel times. Instead, in energy purchasing (Vinyals et al., 2012), agents cooperate intending to increase monetary savings. However, in most applications, agents pursue both altruistic and selfish goals; for example, in ridesharing, an agent may additionally benefit by lowering travel costs associated with fuel consumption. Such problems require some collective intelligence which guides agents on how to behave. Consequently, solutions are designed to answer the following question: "How should collectives be assembled in order to achieve a common objective?" (Cerquides et al., 2014). Fortunately, this question can be answered by formalizing a collective formation problem in the context of combinatorial optimization.

More in particular, this thesis considers the optimization problem of computing the best set of non-overlapping collectives (i.e. subsets) of agents belonging to a universal set A, so as to maximize the total value provided by a domain-specific utility function, e.g., the reduction in terms of cost or CO_2 emissions associated to the arrangement of a shared trip (Bistaffa et al., 2019) or the improvement thanks to cooperation within a team of students (An-

drejczuk et al., 2019).

Formally, let $A = \{a_1, a_2, \ldots, a_n\}$ be a set of n agents and $f : \mathcal{F}(A) \to \mathbb{R}$ a utility function (also referred to as *characteristic function*) that maps every feasible collective[1] $\mathcal{F}(A)$ to a real number. Then, the collective formation problem consists in computing the best set $\mathcal{S}^*$ of non-overlapping subsets of A (also referred to as a *coalition structure* Chalkiadakis, Elkind, and Wooldridge (2011)) that maximizes the sum of the values associated to each collective $S \in \mathcal{S}^*$, i.e.,

$$\mathcal{S}^* = \arg\max_{\mathcal{S} \in \prod(A)} \sum_{S \in \mathcal{S}} f(S), \tag{2.7}$$

where $\prod(A)$ is the set of all partitions of A into non-overlapping feasible subsets. This thesis aims to study the use of machine learning techniques for the formation of collectives. Therefore, the following sections are devoted to reviewing the classical collective formation approaches, i.e., those not including machine learning components. The approaches are presented according to their classification as complete or heuristic solutions. In addition, some complete approaches for coalition formation are known as anytime approaches because of their ability to provide suboptimal solutions before completion, deserving a distinction with respect to purely complete algorithms.

2.2.1 Complete Approaches

Yun Yeh (1986) approach can be visualized on the coalition structure graph 2.1, consisting of partitions (nodes) organized in levels $\Pi_1, \Pi_2, \ldots, \Pi_n$, where level Π_i contains coalition structures with exactly i coalitions. Two nodes in consecutive levels, Π_i and Π_{i+1}, are connected if the coalition structure in Π_{i+1}

[1]Depending on the considered domain, such a set of feasible collectives can be the entire set of subsets of A or, for example, the set of all collectives that satisfy a given constraint (e.g., cardinality constraints (Shehory and Kraus, 1998) or graph-based constraints (Myerson, 1977)).

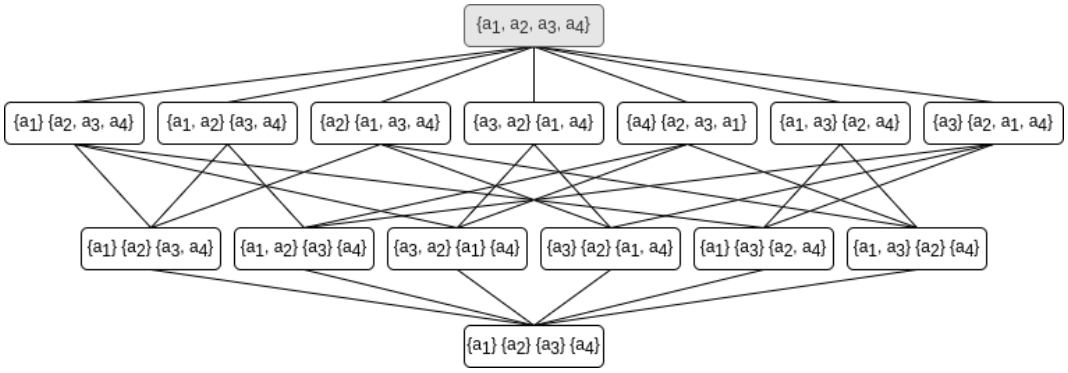


Figure 2.1: Coalition structure Graph.

can be obtained from the one in Π_i by splitting one coalition into two. The *DP* approach evaluates the nodes in one level of the graph and then selects the best move until reaching an optimal node.

One strand of literature (Ieong and Shoham, 2005; Tran-Thanh et al., 2013; Bistaffa, Chalkiadakis, and Farinelli, 2021) has focused on alternative utility function representations, which allow one to reduce the computational complexity of the *CSG* problem by exploiting specific properties of the adopted representation. For example, Ieong and Shoham (2005) proposed a concise representation called *marginal contribution nets*, or MC-nets, where the calculation of the utility is based on a collection of *rules*. Tran-Thanh et al. (2013) proposed a representation called *coalitional skill vector model*, where there is a set of skills in the system, and each agent has a skill vector (a vector consisting of values that reflect the agents' level in different skills). More recently, Bistaffa, Chalkiadakis, and Farinelli (2021) focused on the well-known *induced subgraph game* representation originally introduced by Deng and Papadimitriou (1994) and proposed a *CSG* algorithm based on graph-clustering that exploits the succinctness of the representation.

The approaches presented above avoid an exhaustive search of the space of solutions by exploiting specific properties of the representation or by employing a *DP* strategy. However, a big disadvantage is that, for most of them,

no intermediate solution can be obtained before completion, meaning it is impossible to trade computation time for solution quality. In order to overcome this issue, *anytime* approaches, which can return a solution better than the initial one at any time of the computation, are preferred. Although anytime approaches guarantee optimal solutions, they are rarely used for this purpose because finding an optimal solution in a feasible amount of time becomes impossible because of the complexity of enumerating all possible coalition structures, even for a small number of agents. In the following section, the main anytime solutions for *CSG* are presented.

2.2.2 Anytime Approaches

This section describes the main anytime approaches for the *CSG* problem. According to their solution strategy, these approaches are divided into three categories: identifying subspaces with worst-case guarantees, dynamic programming approaches and integer linear programming approaches.

Identifying Subspaces with Worst-case Guarantees

With the goal of achieving an anytime solution approach for *CSG*, Sandholm et al. (1999), Dang and Jennings (2004), and Rahwan, Michalak, and Jennings (2011) have designed algorithms which divide the search space into disjoint sub-spaces and establish an order in which each sub-space must be searched, ensuring that the worst-case guarantee on the solution quality improves after each sub-space.

While the approaches mentioned above consider divisions of the *coalition structure graph*, other approaches explore other types of search space divisions. For instance, Rahwan et al. (2009) presents an algorithm that searches over the *integer partition graph*. An *integer partition* is a set of positive integers

specifying the size of the coalitions in the subspace. For example, $\{1, 1, 2\}$ is the subspace of all coalition structures composed of two size-one and one size-two coalitions. The *integer partition graph* connects integer partitions which can be obtained by summing two of their integer values, e.g., $\{1, 1, 2\}$ is connected to $\{2, 2\}$ and $\{1, 3\}$. The authors propose a method to evaluate the bounds of each integer partition, thus defining an order in which they must be searched.

Dynamic Programming Approaches

Alternatively, Rahwan et al. (2015) and Michalak et al. (2016) showed that it is also possible to build a dynamic programming anytime algorithm for coalition formation by implementing a *dynamic programming* approach compatible with the *integer partition graph*. To date, this approach is the best anytime algorithm for *CSG* problems, although it cannot be applied to problems with more than 40 agents. These scalability issues manifest the need for approaches that can produce faster solutions which scale to real large-scale problems with hundreds of agents, even if these approaches need to leave behind optimality guarantees or lose on generality by making assumptions about the application domain.

Integer Linear Programming

Different from the previous approaches, the *CSG* problem can be formulated as an *ILP*:

$$\begin{aligned} \text{maximize} \quad & \sum_{S \in \mathcal{F}(A)} f(S) \cdot x_S, \\ \text{subject to} \quad & \sum_{S \in \mathcal{F}(A)} b_{i,S} \cdot x_S = 1, \quad \forall a_i \in A, \end{aligned} \tag{2.8}$$

where x_S is a binary decision variable that encodes whether collective S in the feasible set of coalitions $\mathcal{F}(A)$ is inside the coalition structure $\mathcal{S}$, and $b_{i,S}$ is a

binary value that encodes whether agent $a_i \in A$ belongs to the collective S.

With this formulation, it is possible to apply an *ILP* solver, e.g., *CPLEX* (Cplex, 2009). Although these solvers can also provide a solution before completion, they are highly inefficient compared to previous approaches due to the large number of feasible coalitions that need to be considered in the *ILP* formulation (Rahwan et al., 2015).

Despite its limitations, the *ILP* formulation of the *CSG* problem is essential for this thesis because, when combined with an appropriate reduction of the search space, it is possible to solve the *ILP* in a reasonable amount of time. Some approaches use this technique already, e.g., Bistaffa et al. (2019) formulate the *ILP* with a reduced set of *good coalitions*, proposed by their approach. As a result, they reduce the size of the problem and solve it in a reasonable computation time. Notice that optimality guarantees are lost due to a domain-specific space reduction. Thus, the resulting approach is no longer considered to be anytime but rather a heuristic approach. The following section describes other relevant heuristic approaches following a similar direction.

2.2.3 Heuristic Approaches

Anytime approaches present a solution to the *CSG* problem, which not only provides optimality guarantees on the solutions but can also be applied to any *CSG* problem, regardless of the application domain. Despite these advantages, these approaches cannot be considered for real large-scale problems since they suffer from severe scalability issues. To overcome these, *heuristic approaches* enhance *CSG* by introducing instructions based on assumptions about the application domain. In other words, these approaches trade the generality of complete approaches for an affordable computation time, becoming suitable for real large-scale problems.

Some approaches directly focus on the specific properties of the application

domain. For instance, the work mentioned above by Bistaffa et al. (2019) proposes a solution algorithm for large-scale ridesharing that, by heavily relying on the greedy nature of the domain, is capable of computing solutions of excellent quality for hundreds of agents within one minute. Previously, Farinelli et al. (2017) proposed an approach based on hierarchical clustering that also relies on a greedy heuristic in order to identify the most promising couple of coalitions that can be merged until no beneficial merge can be executed. Unfortunately, these approaches cannot be applied in other *CSG* domains not characterized by such a greedy nature, e.g., the team formation domain discussed in (Andrejczuk et al., 2019).

More recently, Wu and Ramchurn (2020) proposed a *CSG* solution algorithm which defines a search strategy on the coalition structure graph based on *Monte-Carlo Tree Search*. Despite this approach can structurally be applied to any *CSG* problem, it employs a simulation policy based on a greedy heuristic similar to the one proposed by Farinelli et al. (2017). Indeed, the authors report good performance on synthetic datasets that are characterized by a greedy nature.[2]

All these approaches rely on specific properties of the application domain, which is exactly the opposite of the goal of this work, i.e., obtaining a general approach for collective formation problems that do not rely on domain-specific instructions while scaling the number of agents present in real-world problems. Section 2.4 discusses why machine learning is a promising candidate to achieve this goal and reviews the previous attempts integrating machine learning in combinatorial optimization routines.

[2]According to the methodology reported in (Wu and Ramchurn, 2020), the value of a coalition S is correlated with its cardinality $|S|$, hence forming bigger coalitions is, on average, more beneficial.

2.3 Capacitated Multi-Agent Pickup and Delivery

This section presents capacitated *Multi-Agent Pickup and Delivery* (*capacitated MAPD*), a problem involving several agents carrying packages from their pickup to their delivery locations. This problem is relevant for this thesis since it combines a collective formation problem (assigning pickup and delivery tasks to agents) with another combinatorial optimization problem (path-finding for multiple agents). Thus, with minor adjustments, the assignment of tasks can be solved by employing an approach for collective formation. Within this context, the capacitated *MAPD* problem is relevant to this thesis as it allows studying to what extent the proposed approach for collective formation may be used in more complex scenarios and what limitations are confronted.

The following parts of this section provide the definition of the problem and the most relevant literature related to the main techniques useful for solving the capacitated *MAPD* problem.

2.3.1 Problem Definition

The capacitated *MAPD* problem is conformed by a set of agents $A = \{a_1, a_2, \ldots, a_n\}$ which aim at completing a set of tasks $K = \{k_1, k_2, \ldots, k_m\}$. A task k_i consists of moving an item from its pick-up location p_i to its delivery location d_i on a map defined over an undirected graph $G = (V, E)$. Each agent a_i has a capacity C_i, which means that it can execute at most C_i tasks at the same time. Then, the goal is to assign to each agent a_i a subset of tasks $\kappa_i \subseteq K$ and, for each agent, find a conflict-free path $P_i = \langle v_{i0}, v_{i1}, \ldots, v_{ip} \rangle$, such that $v_{it} \in V$, $(v_{it}, v_{it+1}) \in E$. For every task t_j assigned to a_i, its pick-up location must appear previous to its delivery location in the path, i.e., $t < s \;\; \forall v_{it}^{p_j}, v_{is}^{d_j}$,

where the superindex indicates that a given vertex in the path corresponds to a pick-up p_j or a delivery d_j location. Moreover, the load l_{it} of an agent a_i, calculated as the number of picked-up items not yet delivered at timestep t, should never exceed the capacity, i.e., $l_{it} \leq C_i \ \ \forall t$. A path is said to be conflict-free if:

(i) No vertex is occupied by more than one agent at the same timestep,

$$v_{it} \neq v_{jt} \ \ \forall a_i, a_j, t. \tag{2.9}$$

(ii) No edge is traversed by more than one agent at the same timestep,

$$(v_{it}, v_{it+1}) \neq (v_{jt+1}, v_{jt}) \ \ \forall a_i, a_j, t. \tag{2.10}$$

The goal of the *capacitated MAPD* is finding an assignment whose agents, following optimal conflict-free paths, minimize the *total travel distance*, i.e. the sum of path lengths over all agents,

$$TTD = \sum_{a_i} |P_i|. \tag{2.11}$$

Similar to the capacitated *MAPD* problem, there exist other problems which aim at planning paths for multiple agents. Such problems are known as *Multi-Agent Path Finding* (*MAPF*), and many of the techniques applied for it are also useful for the capacitated *MAPD*. Following, the widely adopted *Conflict-Based Search* approach for conflict resolution in *MAPF* is presented, along with some of its most popular variants which are employed later on as part of the proposed methodology.

2.3.2 Conflict-Based Search and Variants

Conflict-Based Search (*CBS*) (Sharon et al., 2015) is a prominent algorithm for optimally solving *MAPF* problems. CBS relies on a bi-level search to resolve conflicts by adding constraints at the higher level and replanning paths for agents that respect these constraints at the lower level. The higher level of CBS performs a best-first search on a binary search tree known as the constraint tree. A node N in the constraint tree contains the following information:

1. $N_{constraints}$ is the set of constraints imposed thus far in the search. There are two types of constraints: i) vertex constraints derived from violating equation 2.9; and ii) edge constraints derived from violating equation 2.10.

2. $N_{solution}$ is a set of individual optimal paths for each agent while considering the constraints in $N_{constraints}$. The optimal paths are found via a low-lever search strategy such as the *space-time* $A*$ algorithm (Silver, 2005), a variation of $A*$ introduced to take constraints into account.

3. N_{cost} is the sum of costs of individual optimal paths in $N_{solution}$.

4. $N_{conflicts}$ is the set of conflicts between any two paths in $N_{solution}$.

At a higher level, *CBS* starts with a constraint tree with only one node whose set of constraints is empty. The constraint tree is continuously expanded in a best-first fashion, always expanding the node with the lowest N_{cost}. After selecting a node to expand, *CBS* checks the conflicts in $N_{conflicts}$ and, if none exist, *CBS* terminates and returns $N_{solution}$. Otherwise, *CBS* chooses one of the conflicts at random and adds two child nodes to N. Each child node inherits $N_{constraints}$ from the parent and adds a constraint for one of the conflicting agents. Then, employing the low-level path-finding strategy, the

optimal path of the agent affected by the new constraint is recomputed, and N_{cost} and $N_{conflicts}$ are updated. *CBS* guarantees an optimal solution because it explores the constraint tree in a best-first manner, only avoiding expanding tree nodes with a lower bound solution cost better than the current best solution cost.

Several approaches have been devised upon *CBS*, mainly improving on the random conflict-selection strategy adopted by the original version. For instance, Boyarski et al. (2015) introduced *Improved CBS* (*ICBS*) which classifies conflicts into three types in order to prioritize them: when expanding N into two child nodes, (i) *cardinal* conflicts derive into two child nodes with costs higher than N_{cost}; (ii) *semi-cardinal* conflicts derive into one child node with cost higher than N_{cost}; and, (iii) *non-cardinal* conflicts derive into no child node with cost higher than N_{cost}. The cost of child nodes can be determined without expanding a node by employing *Multi-Valued Decision Diagrams*. *ICBS* is able to improve its efficiency by first resolving cardinal conflicts, then semi-cardinal conflicts and finally non-cardinal conflicts since it increases the lower bound on the optimal cost faster by generating child nodes with higher costs.

Barer et al. (2014) presented *Greedy CBS*, a sub-optimal variant of *CBS*. The authors propose a relaxation of the high-level strategy by introducing five heuristics for selecting nodes to expand. These are the number of conflicts, the number of conflicting agents, the number of pairs of agents that have at least one conflict between them, a vertex cover of conflicts and an alternating heuristic (Röger and Helmert, 2010). Finally, Ma et al. (2019) proposed *PBS*, an approach that incorporates prioritized planning into *CBS*. *PBS* establishes a priority in which agents must plan their paths. Then, the path of each agent is planned based on their priority using a low-level strategy that only considers conflicts with previously planned agent paths. Although the approach does not guarantee finding optimal solutions, the authors show state-of-the-art solution

quality and runtimes when compared to other sub-optimal approaches.

CBS and its variants are a general approach for conflict resolution in the context of *Multi-Agent Path Finding*. However, *CBS* alone does not capture the nature of many real-world problems, where agents able to plan for executing multiple tasks are constantly engaged with new tasks. The following section discusses approaches which expand the limits of *CBS* by incorporating a task assignment component.

2.3.3 Integrated Task Assignment and Conflict Based Search

Ma et al. (2017) consider a stream of tasks that arrive to the agents in an online setting. The authors introduce the *token-passing* approach, which first assigns tasks to agents with the *Hungarian method* and then employs *CBS* to plan conflict-free paths. Similar to the problem formulation considered in this thesis, Liu et al. (2019) consider an offline version of the problem, which assumes all incoming tasks are known in advance and agents can carry more than one task. The authors propose *TA-Hybrid*, which first solves a *TSP* formulation of the task assignment problem with an existing *TSP* solver and then plans the conflict-free paths employing *CBS*.

Previous approaches perform the task assignment and planning of conflict-free paths separately. Thus, assignments are performed not considering the increment in the path cost caused by conflicts appearing during the conflict resolution strategy. To address this, Chen et al. (2021) propose an approach integrating *PBS* into the assignment of tasks. To this end, their approach continuously updates a priority heap determining which task is going to be assigned to which agent based on a *regret-based marginal-cost assignment* (*RMCA*).

This thesis proposes an integrated solution detailed in chapter 4, similar to the one by Chen et al. (2021), incorporating an *attention-based* model to learn the priority heap and guide the task assignment process.

2.4 Machine Learning for Combinatorial Optimization

The use of machine learning techniques to solve combinatorial optimization problems is a recent yet very active topic that has received significant attention during the last few years (Bengio, Lodi, and Prouvost, 2021; Kotary et al., 2021). According to Bengio, Lodi, and Prouvost (2021), machine learning can contribute to the optimization field in two ways: i) replace some heavy computations by learning fast approximations without deriving new explicit algorithms, and ii) improve the optimization approach by deriving a policy learning from the domain-specific structure. In addition, learning can be useful to generalize across a class of different combinatorial optimization problems with similar structures. The objective of this thesis is the study of machine learning solutions to enhance the performance of classical combinatorial optimization solvers by learning the domain-specific structure. Moreover, generalization across a class of different problems can be achieved, given that learning can be performed offline for a given domain before the deployment of the solution. Therefore, because of its practical interest, the generalization capability of the approaches is a crucial aspect also studied in this work.

The above objectives result in key challenges for machine learning approaches. The following sections provide a detailed discussion of the main learning challenges of combining machine learning and combinatorial optimization: learning algorithmic decisions, learning to generalize, and learning problem constraints. Following that, a literature review of the recent approaches in the field will be presented.

2.4.1 Learning Algorithmic Decisions

Following the previous discussion, machine learning seeks to substitute a set of instructions that comprise a part or the totality of an algorithmic solution. To this end, the machine learning component must improve its decisions by learning the ones producing satisfactory outcomes. While *Reinforcement Learning* improves the decisions by optimizing their observed outcome, *Supervised Learning* copies the decisions made by a benchmark algorithm. These frameworks will be discussed further below as emerging solutions for learning decisions. The discussion stems from the mathematical formulation of learning algorithmic decisions inspired by the one from Bischl et al. (2016) and Bengio, Lodi, and Prouvost (2021).

Given a set of algorithms $\mathcal{A}$, one might ask which is the best one for a problem from which several instances $\mathcal{I}$ are observed, each with probability $\mathcal{P}$. The performance of an algorithm is measured by a function $m : \mathcal{I} \times \mathcal{A} \to \mathbb{R}$ which traditionally is the cost of a solution but can also incorporate other information, e.g., the running time or variety of solutions. Assuming that lower m is better, one would say that the best algorithm is the one which satisfies

$$\min_{a \in \mathcal{A}} \mathbb{E}_{i \sim \mathcal{P}} m(i, a). \tag{2.12}$$

Since it is not practical to compute the expected performance by considering the average over all instances, a dataset with a reduced number of instances is used to estimate these quantities:

$$\min_{a \in \mathcal{A}} \frac{1}{|D|} \sum_{i \in D} m(i, a). \tag{2.13}$$

As a deep learning approach, neural networks can behave as algorithms without including specific instructions other than several algebraic operations

between its parameters $\theta \in \mathbb{R}^p$. In that sense, instead of searching for the best algorithm in a finite set, one can search over a continuous space of parameters,

$$\min_{\theta \in \mathbb{R}^p} \frac{1}{|D|} \sum_{i \in D} m(i, a_\theta). \tag{2.14}$$

Since the parameters θ are defined over a continuous space, gradient methods are handy for solving the learning task. Nevertheless, the performance measure is usually not differentiable, meaning that equation 2.14 needs to be manipulated in order to apply gradient methods. One possibility is to substitute the performance measure from the previous equation with a loss function $\mathcal{L}_i(a^*, a_\theta)$ representing the distance between the neural network outputs a_θ and the expert algorithm ones a^*. The optimization goal then becomes minimizing the distance between decisions until the neural network eventually mimics the expert, resulting in a *supervised learning* approach. It is worth noting that, in this case, the cost function m is removed from the equation, becoming a viable approach for domains where this function is unknown, does not align with the real optimization goal, or is prohibitively expensive to compute during training. Still, a significant disadvantage is that the performance of the learned decisions is bounded to the expert algorithm, which quality might be unsatisfactory.

Another problem to consider is the presence of different sources of randomness, e.g., non-deterministic domains and models. In this situation, one might also need to estimate the performance measure accounting for a source of randomness τ,

$$\min_{\theta \in \mathbb{R}^p} \frac{1}{|D|} \sum_{i \in D} \mathbb{E}_\tau \left[m(i, a_\theta) \right]. \tag{2.15}$$

A commonly used framework when learning over a stochastic domain with sequential decisions is a *Markov Decision Process*, which models the dynamics

of the environment as stochastic transitions over a state space. A straightforward solution approach to this framework is the one of *RL*. In this framework, a stochastic policy π_θ, making algorithmic decisions, is optimized directly by adjusting the probabilities to those showing a better performance measure (expected reward in the *MDP* formulation).

In conclusion, one may employ a *supervised learning* approach imitating algorithmic decisions which replace heavy computation with a fast approximation. Furthermore, if one is interested in discovering new strategies, the *reinforcement learning* framework can learn algorithmic decisions solely through experience.

To this point, the learning instances have been assumed to define the learning problem entirely. In practice, since not all instances from $\mathcal{P}$ will be observed during training, the model may learn a behaviour specific to the training instances. Moreover, some instances requiring an effective policy may be underrepresented in $\mathcal{P}$ or not represented at all. While reinforcement learning approaches address this issue by employing a stochastic policy to enhance exploration of the search space, supervised learning approaches need to consider training instances carefully crafted to facilitate the learning task (Kotary, Fioretto, and Van Hentenryck, 2021). The ability of an approach to perform well on unseen instances is referred to as its generalization ability. This concept is expanded in the following section, and the methods for achieving various forms of generalization are described.

2.4.2 Learning to Generalize

The term generalization can usually refer to a variety of things; this thesis considers generalization to i) unseen instances, ii) a reparameterized problem, and iii) different problems.

When training an algorithm with equation 2.14, the performance measure

is optimized only considering the instances in the dataset. In reality, one would like an algorithm that performs well on any problem instance. To assess the capacity of a model to learn decisions for instances other than the ones in the dataset, it is a widespread procedure to create a test dataset with instances unseen during training and evaluate the performance measure improvement on this one (Raschka, 2018).

On an upper generalization level, the objective is abstracting the logic behind solving one problem to different reparametrizations of the same problem. In this situation, instances may be similar, if not identical, but the problem may have various parameterizations or include new constraints. For example, consider a model trained for the capacitated VRP; one could change the maximum vehicle capacity or set a different scenario, such as another city. In this case, it is unlikely that good performance on the reparameterized problem can be obtained without retraining the model. Nevertheless, this is not a problem in practice since training can be done beforehand, and this has no impact on the computation time of the learned algorithmic approach used to solve the problem. Moreover, the reparameterized problem shares its structure with the original one; thus, most algorithmic decisions will not change from one approach to the other. Therefore, it is unnecessary to retrain the neural network entirely. Instead, some of its parameters can be fixed, only retraining some following a *transfer learning* approach (Weiss, Khoshgoftaar, and Wang, 2016).

The highest level of generalization considered in this thesis is the knowledge abstraction to different problems. The difficulty in this scenario stems from the diverse structures found in different problems. Routing problems, for example, assemble a sequence of waypoints, whereas set partitioning constructs a set of partitions, or one problem may consider continuous variables while another only accepts integer ones. Moreover, since the problem struc-

ture is entirely different, the underlying logic behind algorithmic decisions also changes. Therefore, applying any transfer learning approach as in the previous case would probably fail, and the only option is to retrain a model for the new problem. As discussed in the upcoming chapter, this thesis proposes a general approach for the formation of collectives, a set of problems with a shared structure, allowing for a common solution generalizing over this class of problems. The following section presents the last challenge of combining machine learning and combinatorial optimization: learning problem constraints.

2.4.3 Learning Constraints

One core challenge when employing a machine learning approach for learning strategies to solve combinatorial optimization problems is learning to fulfil the hard constraints inherent to the problem. Hopfield and Tank (1985) proposed Hopfield Networks with a modified energy function to emulate the loss function of a TSP and employed Lagrange multipliers to penalize the violation of problem constraints. When carefully benchmarked, Hopfield Networks have not yielded satisfying results (La Maire and Mladenov, 2012; Sarwar and Bhatti, 2012). As a result, these approaches have fallen in popularity in favour of approaches forestalling the problem. Most current methods either i) hard-encode the constraints into the network structure, e.g., by setting the input and output network size to the constrained one or ii) employ a classical solver such as an ILP to encode the constraints not considered by the learning approach. In this work, a combination of both is considered, as described in the next chapter.

However, previous practices do not provide the network with information about the constraints it encodes, so these are never learned. Neural networks being agnostic about the constraints is a limiting factor in the design of general machine learning approaches for combinatorial optimization since the network

structure must be explicitly hand-tuned for each problem constraint. Recently, some approaches employ the Lagrangian duality to introduce problem constraints into the learning objective of problems such as the *Optimal Power Flow* (Fioretto, Mak, and Van Hentenryck, 2020; Chatzos et al., 2020), transprecision computing (Fioretto et al., 2021) and fair classification (Tran, Fioretto, and Van Hentenryck, 2021). In the context of this thesis, constraints can be hard-encoded into the architecture because they are well-defined properties of the domain, e.g., the size of collectives. However, introducing constraints with the previous techniques is an interesting direction to increase the generality of the proposed approach.

2.4.4 Prominent Machine Learning Approaches for Combinatorial Optimization

This section describes the main approaches combining machine learning and combinatorial optimization. Because there are so many existing solutions, they are categorized adopting Bengio, Lodi, and Prouvost (2021)'s classification along two axes: algorithmic structure and learning methodology.

Along the first axis, depending on the structure of the overall approach, Bengio, Lodi, and Prouvost (2021) identifies two alternatives: i) end-to-end machine learning approaches which are able to directly construct a solution for optimization problems, and ii) mixed approaches which use machine learning as a subroutine of a classical optimization approach, either as a preprocessing step or used alongside the classical approach in an online scheme. The second axis considers the learning methodology used, i.e., supervised, unsupervised, or reinforcement learning. This section presents machine learning approaches for combinatorial optimization based on this classification.

Algorithmic Structure

Examining machine learning approaches for combinatorial optimization from its algorithmic structure, these can be classified either as end-to-end approaches or mixed approaches.

End-to-End Approaches - A wide variety of end-to-end approaches are developed to bring approximate and fast solutions to combinatorial optimization problems. For instance, Vinyals, Fortunato, and Jaitly (2015) propose the *pointer network* architecture, an encoder-decoder architecture consisting of a combination of recurrent neural networks with an attention operation to solve the problem of previous sequence-to-sequence models unable to produce variable size output sequences. This new architecture enables the generation of permutations of the input sequence, providing a practical use case of their approach to generating convex hulls, Delaunay triangulations, and TSP solutions. Khalil et al. (2017) propose a graph embedding network to learn combinatorial optimization algorithms over graphs, managing to learn a solution algorithm for *Minimum Vertex Cover*, *Maximum Cut* and the TSP. In the same direction, Joshi, Laurent, and Bresson (2019) proposes Graph Convolutional Networks to improve the solution quality for the TSP. However, because their approach fails to generalize to variable problem sizes, the authors suggest that training in a reinforcement learning setting, rather than a supervised learning one, could address this issue. Motivated by the success in machine translation of pure *attention-based* architectures (Vaswani et al., 2017), Kool, Van Hoof, and Welling (2018) proposed a similar encoder-decoder approach for the *TSP*, *VRP*, *Orienteeing Problem* and *Stochastic Prize-Collecting TSP*, only employing attention operations in the network. In addition to the previous approaches, other end-to-end approaches have been devised for variants of the *VRP*, such as the capacitated *VRP* (Nazari et al., 2018) or the online capacitated *VRP* (James, Yu, and Gu, 2019).

Another strand of literature focuses on designing combinatorial optimization neural layers to enhance learning combinatorial decisions. Amos and Kolter (2017) started this line of research by providing differentiable KKT conditions, which solve a Quadratic Program optimally. Donti, Amos, and Kolter (2017) showed that neural layers could also be trained to learn decisions in stochastic domains such as an inventory stock problem, a real-world energy grid scheduling task and a real-world energy storage arbitrage task. Other notable approaches propose similar solutions for problems with linear objectives (Elmachtoub and Grigas, 2022) and satisfiability problems (Wang et al., 2019).

Mixed Approaches - Mixed approaches aim at combining machine learning with classic optimization procedures. In this line of research, a strand of literature focuses on using machine learning as a preprocessing step before a classical optimization approach. In particular, the approach proposed by Ding et al. (2020) solves eight different classical problems, including *TSP* and *VRP*, employing Graph Convolutional Neural Networks which predict the value of binary variables in a *MILP* formulation of these problems. The solution is then found by means of a branch and bound approach, which uses the values to guide the search. In the same way, Li, Chen, and Koltun (2018) tackle benchmark satisfiability problems related to social network graphs utilizing Graph Convolutional Neural Networks to select the nodes in a graph that are likely to appear in an optimal solution and then solve the problem for the reduced graph. Similar to neural networks, *Support Vector Machines* can also play the role of a heuristic approach as a preprocessing step for optimization problems. For instance, the work of Xavier, Qiu, and Ahmed (2021) uses *k-Nearest Neighbors* in addition to a *Support Vector Machine* to significantly reduce the problem size for Security-Constrained Unit Commitment problems in power systems and electricity markets. Another example is the work of

Sun, Li, and Ernst (2019), which uses a *Support Vector Machine* to find a graph reduction for the maximum weight clique problem. Many of the approaches which use machine learning as a preprocessing step focus on reducing the optimization problem (i.e., making the problem smaller and computationally tractable). Although problem reduction with machine learning does not provide any optimality guarantee, these approaches can usually provide high-quality solutions due to the "backbone" structure of the optimization problems they tackle (according to the terminology adopted by (Kilby, Slaney, Walsh, et al., 2005)), i.e., optimal solution and high-quality solutions are likely to share a specific structure. In contrast to approaches using machine learning as a preprocessing step, some mixed approaches aim to incorporate machine learning into well-established optimization approaches. For example, Hottung and Tierney (2019) use an *attention-based* model in order to reconstruct solutions inside a Large Neighborhood Search setting for the Capacitated *VRP*. Another example is the work of Hottung and Tierney (2019), in which neural networks are used as a heuristic to guide search inside a tree search approach for the *container pre-marshalling problem*.

This thesis employs a mixed approach combining a machine learning component and an *ILP* solver. In contrast to end-to-end approaches, this combination allows the machine learning component to be agnostic about the constraints and focus on generating high-quality collectives. The *ILP* solver considers the generated collectives and assembles a solution composed of disjoint collectives.

Learning Methodology

According to the learning methodology, approaches are classified either as learning from demonstration (*supervised learning*) or learning from experience (*reinforcement learning*). Unsupervised approaches are not discussed since

their application to combinatorial optimization is scarce.

Supervised Learning - Several approaches adopt supervised learning strategies with the objective of imitating the outputs of an already existing solution algorithm, hence, by following Bengio, Lodi, and Prouvost (2021)'s terminology, "replacing it by its ML approximation". For example, the work by Li, Chen, and Koltun (2018) makes use of supervised learning to train Graph Convolutional Neural Networks for Maximal Independent Set, Minimum Vertex Cover, Maximal Clique, and SAT. Khalil et al. (2016) propose an approach to learn from a precomputed dataset a ranking function which is then used as a branching heuristic for *MILP*. Gasse et al. (2019) and Nair et al. (2020) also aim at constructing a branching heuristic for solving *MILP*, but in their case, they directly learn to imitate the decisions made by an expert by means of a cross-entropy loss. Most of these approaches aim to select an option among several alternatives, e.g., variable selection in branch-and-bound approaches. Another alternative is using machine learning to decide if a computationally demanding sub-routine needs to be performed. Following this direction, the work by Kruber, Lübbecke, and Parmentier (2017) uses a supervised learning approach to determine whether a Dantzig-Wolfe decomposition should be applied to a *MILP* instance in order to solve it faster.

Reinforcement Learning - Alternatively, reinforcement learning aims at learning a policy (i.e., a system which determines a course of action) which optimizes the sum of future expected rewards observed from the actions the policy performs. The policy is usually modelled inside a *Markov Decision Process* and interacts with the environment by executing an action and receiving an observation and a reward signal. The use of reinforcement learning is appealing in the context of mixing machine learning with combinatorial optimization because classical algorithms used for this purpose usually involve sequential decisions, which can be modelled as a *Markov Decision Process*. Moreover,

in contrast with supervised learning, reinforcement learning does not require other approaches to learn from since it learns purely from experience. Several approaches aim to replace supervised learning with reinforcement learning in the context of combinatorial optimization. For example, the work by Bello et al. (2016) and the work by Nazari et al. (2018) propose reinforcement learning as an alternative way to train Pointer Networks, previously trained with supervised learning (Vinyals, Fortunato, and Jaitly, 2015). Another example is the work by Kool, Van Hoof, and Welling (2018), which also uses reinforcement learning to train an *attention-based* model for the *TSP* and the *VRP*.

In this work, *supervised learning* is initially considered to train the machine learning component learning to generate collectives. However, it is abandoned because it limits the performance of the machine learning model to the training examples obtained from a benchmark approach. As a result, it is replaced by *reinforcement learning*, which does not require training examples as it learns solely through experience.

2.4.5 Prominent Machine Learning Approaches for *MAPF*

Machine learning has received little attention from the *MAPF* community. The first attempt to incorporate machine learning into existing optimization-based solutions was by Sartoretti et al. (2019), who proposed a combination of reinforcement and supervised learning to learn decentralized policies for agents aiming to find the shortest path while avoiding conflicts in a large factory-like environment. Later, Damani et al. (2021) extend the work by Sartoretti et al. (2019) to the *capacitated MAPD* problem, achieving better performance on more dense environments populated with up to 2048 agents.

Huang, Koenig, and Dilkina (2021) propose a machine learning framework for conflict selection that learns the decisions made by a linear ranking function inspired in Khalil et al. (2016). Similarly, Huang, Dilkina, and Koenig

(2021) combine supervised learning and curriculum learning to learn a node-selection strategy to accelerate a variant of *CBS*. In Huang et al. (2022), the authors propose an anytime solution that uses machine learning to learn how to choose a subset of agents from a collection of subsets, such that replanning will increase the solution cost the most. Zhang et al. (2022) propose a machine learning framework to learn a good priority ordering, similar to the one in Ma et al. (2019). Instead, this thesis adapts the machine learning approach for collective formation to perform the task assignment in the capacitated version of the *MAPD* problem.

Chapter 3

Machine Learning for Collective Formation

There exist several contemporary applications involving the formation of collectives, such as ridesharing and team formation, which are studied in this thesis. Also, due to an increasingly interconnected society, new problems involving the formation of collectives are emerging and becoming more relevant in the last decades. Therefore, general-purpose solvers able to produce high-quality solutions are required to tackle these emerging problems efficiently. In this direction, this thesis aims to study machine learning as a possible approach to address the lack of generality of current ad-hoc approaches for collective formation problems.

In the previous chapter, a variety of combinatorial optimization problems were discussed, along with the challenges and limitations of current solution approaches. One of the most critical challenges is designing algorithmic solutions that are fast, general and exact. For real-world applications, which typically consist of hundreds of variables, fast solutions can only be obtained by sacrificing exact solutions in favour of approximate heuristics. Furthermore, these heuristics typically include domain-specific instructions to improve their

performance in these large-scale settings, resulting in more ad-hoc and less general approaches.

Along these lines, machine learning has gained popularity as a promising approach to achieve fast and general solutions to combinatorial optimization tasks, in some cases even improving the state-of-the-art. Within the purview of the literature review conducted for this thesis, the solutions presented in the previous chapter are the vanguard employing machine learning for combinatorial optimization tasks. Still, none applies to collective formation. The following section proposes a solution approach incorporating machine learning into a general-purpose solver for collective formation.

3.1 Solution Approach

Collective formation is a combinatorial problem in the sense that forming one collective influences the possibility of forming others. Consequently, an optimal solution is likely to be formed by collectives that are not only high-quality but also do not constrain the formation of other high-quality ones. Therefore, an end-to-end approach aiming for optimal solutions must ponder collectives from these two different perspectives, i.e., quality and constraints. The solution approach proposed in this thesis attacks this dual perspective by combining machine learning and a classical solution approach. On the one hand, machine learning excels at learning the inherent structure of high-quality collectives, although it does not successfully capture the combinatorial influence these exert on one another. On the other hand, a classical approach is used to resolve the constraints imposed by the combinatorial nature of the problem, which the machine learning approach does not take into account.

In this direction, the proposed approach considers the *ILP* formulation of the coalition formation problem described in equation 2.8. As discussed

previously, an *ILP* cannot be fully solved in a reasonable time because of the intractable number of possible coalitions appearing in large-scale *CSG* problems (Rahwan and Jennings, 2008). This thesis proposes a technique to reduce the number of coalitions that need to be considered by the *ILP*. More precisely, by learning the inherent structure of the domain, a machine learning component proposes a reduced set of *promising* collectives $\mathcal{R}(A)$, from which an *ILP* solver computes a suboptimal high-quality solution from the *ILP* formulation:

$$\begin{aligned} &\text{maximize} && \sum_{S \in \mathcal{R}(A)} f(S) \cdot x_S, \\ &\text{subject to} && \sum_{S \in \mathcal{R}(A)} b_{i,S} \cdot x_S \leq 1, \quad \forall a_i \in A. \end{aligned} \tag{3.1}$$

As depicted in figure 3.1, the approach is divided into two separate steps. First, the machine learning component receives the agents and proposes a reduced set of candidate collectives $\mathcal{R}(A)$. Then, equation 3.1 is formulated with the collectives $S \in \mathcal{R}(A)$ and solved to obtain the collective formation solution.

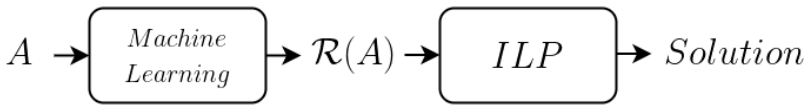


Figure 3.1: General approach for collective formation. The machine learning component generates a reduced set of collectives from which an *ILP* solver computes the solution.

As previously stated, an optimal solution is made up of lower-quality collectives in addition to higher-quality ones. As a result, the machine learning component, despite being trained to generate high-quality collectives, should also provide lower-quality alternatives. One possible way to achieve this is by considering a model defining a probability distribution over collectives, from which diverse collectives can be sampled. This technique is employed by sequential decision models such as *pointer networks* and *attention-based* approaches. Another possibility is using a random latent variable, as in generative models such

as *generative adversarial networks*, which introduce a source of variance within the models. The following sections discuss these approaches to implement the machine learning component in figure 3.1 to generate collectives.

3.2 Generative Adversarial Networks

Generative Adversarial Networks (*GANs*) are a generative model composed of two neural networks, a generator and a discriminator, competing with one another (Goodfellow et al., 2014). The task of the generator network (parametrized by $\boldsymbol{\theta}^g$) is, given a random latent vector of inputs $\boldsymbol{z}$, to generate an output sample $\boldsymbol{x} = g(\boldsymbol{z}; \boldsymbol{\theta}^g)$ that resembles a sample from a probability distribution p_{data}. The task of the discriminator d (parametrized by $\boldsymbol{\theta}^d$) is, given a sample $\boldsymbol{x}$, to classify it as *real* if it belongs to the data distribution $\boldsymbol{x} \sim p_{data}$, or *fake* if it belongs to the generator distribution $\boldsymbol{x} \sim p_{g(\boldsymbol{z})}$.

The two networks are trained simultaneously with adversarial goals setting a two-player mini-max game. The discriminator, presented with a binary classification task (fake vs real samples), is trained to minimize a binary cross-entropy loss $V(d, g)$. The generator, aiming at generating realistic samples which can fool the discriminator, is trained to maximize the same loss. During training, each network tries to optimize its objective, resulting in

$$\min_g \max_d \left[V(d,g) = \mathbb{E}_{\boldsymbol{x} \sim p_{data}} \left[\log d(\boldsymbol{x}) \right] + \mathbb{E}_{\boldsymbol{x} \sim p_{g(\boldsymbol{z})}} \left[\log(1 - d(g(\boldsymbol{z}))) \right] \right] . \quad (3.2)$$

Conveniently, only the right summand in equation 3.2 depends on the generator parameters $\boldsymbol{\theta}^g$. Thus, considering only this term in the loss does not change the gradients for the generator. Hence, as pictured in figure 3.2, it is enough for the generator to minimize $\mathbb{E}_{\boldsymbol{x} \sim p_{g(\boldsymbol{z})}} \log(1 - d(g(\boldsymbol{z})))$, which is the same as maximizing $\mathbb{E}_{\boldsymbol{x} \sim p_{g(\boldsymbol{z})}} \log d(g(\boldsymbol{z}))$.

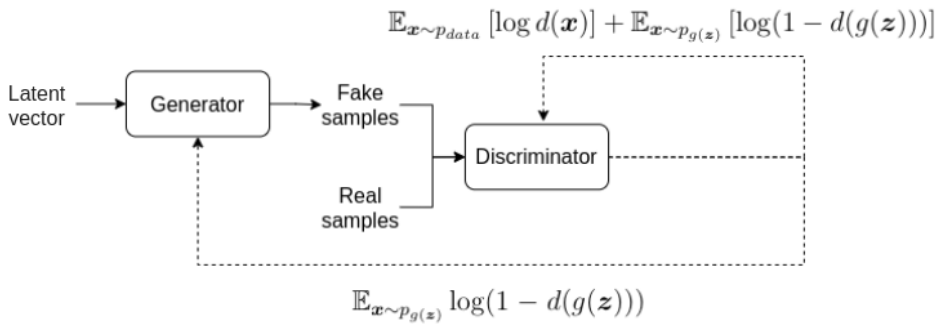


Figure 3.2: Schema of the Generative Adversarial Networks training. The generator network receives a latent vector and produces a fake sample. The discriminator receives a combination of real and fake samples and classifies them according to their origin. The loss is computed with the predictions of the discriminator, and the gradients are propagated to each parameter of the network.

Simultaneously training both networks in a competitive scenario makes training *GANs* a non-trivial task. One of the most concerning issues is that the loss function may not converge due to oscillations of the network parameters, i.e., the generator and discriminator may adapt to each other repeatedly without showing any long-term improvement. Furthermore, the discriminator might learn faster than the generator, thus classifying almost every sample produced by the generator as fake. Therefore, the generator cannot learn to generate samples fooling the discriminator, resulting in a training cul-de-sac. The most recurrent *GANs* training problem is mode collapse, i.e., the generator achieves to deceive the discriminator systematically with a limited variety of samples.

After discussing the model and the common problem derived from its complex training setup, the following section describes how *GANs* can be used to generate collectives.

3.2.1 Generating Collectives

The proposed general approach for collective formation employs a machine learning component in order to produce a set of promising collectives of high value. Thus, the role of the *GAN* is to train a generator to sample such collectives. To this end, the set of *real* samples, which the *GAN* learns to imitate, needs to be composed of high-quality collectives. In order to generate a dataset of collectives of high value, one has two options: i) generate collectives employing another expert algorithm producing acceptable solutions, or ii) generate collectives randomly and select those with a characteristic function value over a certain threshold. The first option is better if one has access to an expert algorithm, but it also caps the collective values produced by the generator. The second option is preferable if no acceptable algorithm is available or if bounding the solutions to those of an external algorithm is not desired. This thesis aims to propose a general approach that does not rely on any external algorithm to generate high-value collectives; hence the second option is preferred.

When considering the formation of collectives, the structure of the neural network differs in some aspects from the *GAN* proposed by Goodfellow et al. (2014). Since the generated collectives are conditioned on the agents present in one collective formation instance, the generator should receive them as an input in addition to the latent vector. Furthermore, because agents and collectives are variable-size structures, the generator must accept variable-size inputs and outputs. Although *GANs* were originally proposed as *Multi-Layer Perceptron* networks, some variants admit sequential data of variable size, e.g., a generator employing *Recurrent Neural Networks* (Mogren, 2016), and an *attention-based GAN* (Xu et al., 2018). Still, this thesis considers a *Multi-Layer Perceptron GAN* to be the most direct approach for collective formation. Consequently, the problem of variable-size inputs and outputs is addressed by encoding these

in a domain-specific size-invariant representation.

In conclusion, one can include variable-size structures with the appropriate *GAN* architecture. However, despite not employing a traditional *supervised learning* training setup, *GANs* present similar data generation challenges to those discussed in Kotary, Fioretto, and Van Hentenryck (2021) for supervised approaches. In practice, bounding the approach to the solutions provided by an expert is not desired for a general approach. Alternatively, generating a sufficient number of training instances through random sampling and filtering out the undesirable ones is not practical and results in low-quality solutions. Therefore, the following sections present *pointer networks* and *attention-based* models, two sequence-to-sequence approaches with prominent applications to combinatorial optimization. Both approaches allow for a reinforcement learning training setup that does not require examples from which to learn.

3.3 Pointer Networks

Vinyals, Fortunato, and Jaitly (2015) proposed *Pointer Networks*, a sequence-to-sequence approach for learning the conditional probability of an output sequence whose elements are discrete tokens corresponding to positions in a variable-size input sequence. The model consists of two *recurrent neural networks* performing as an encoder and a decoder. The encoder network processes an input sequence S of size n, one element at a time, and produces a sequence of hidden memory states $\{enc_i\}_{i=1}^{n}$, where $enc_i \in \mathbb{R}^d$, being d the dimensionality of the hidden states. At each decoding step, the decoder produces a hidden memory state dec_j, which is combined with the encoder hidden memory states $\{enc_i\}_{i=0}^{n}$ to compute the probability of selecting the input s_i in the output

position j via an attention operation (Bahdanau, Cho, and Bengio, 2014):

$$u_i^j = v^T \tanh\left(W_{enc} enc_i + W_{dec} dec_j\right), \tag{3.3a}$$

$$p_i^j = softmax(C \tanh\left(u_i^j\right)), \tag{3.3b}$$

where $C \tanh\left(u_i^j\right)$ clips the values of $u_i^j \in \mathbb{R}$ in the range $[-C, C]$, to control the entropy of the probabilities p_i^j. The $W_{enc}, W_{dec} \in \mathbb{R}^{d \times d}$ and $v^T \in \mathbb{R}^d$ are learnable model parameters. Notice how $\{enc_i\}_{i=0}^n$ includes an additional token $enc_0 = \langle eos \rangle$ that can be selected at decoding time to complete the process. In addition, a mask is applied to avoid sampling an input element twice, i.e., $u_i^j = 0$ if input element s_i has been selected at $j' < j$. Figure

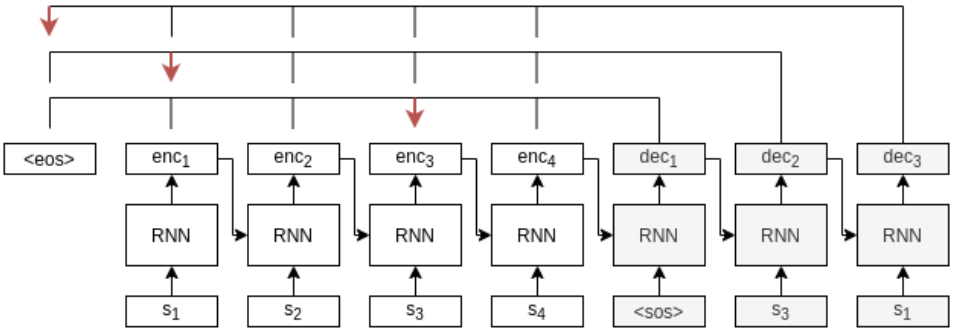


Figure 3.3: *Pointer Network* decoding example. White boxes represent the encoder inputs, *RNN* cells and hidden states. Grey boxes represent the decoder inputs, *RNN* cells and hidden states. The arrows at the top of the figure represent the output of the attention operation.

3.3 exemplifies the *Pointer Network* decoding process. The encoder (white) processes a sequence with four input elements producing the same number of hidden states. The decoder (grey) receives the last encoder hidden state and a start-of-sequence $\langle sos \rangle$ token, producing a decoder hidden state used along an attention operation to select input element s_3. Then, s_3 and the current hidden state are the inputs of the next decoding iteration, selecting s_1. In the third decoding iteration, the attention operation selects the end-of-sequence

$\langle eos \rangle$ token, finishing the decoding process.

The following section describes how the *Pointer Networks* architecture is used for generating collectives.

3.3.1 Generating Collectives

Because the *Pointer Network* is a sequence-to-sequence model, it is ideal for processing variable-size structures such as collective formation instances. In this regard, the encoder processes the agents one by one, storing each agent representation in the encoder memory states. The decoder then produces a collective by sampling one agent at a time from the probabilities computed by the attention operation, which takes into account the hidden memory states of the encoder and decoder. Despite admitting variable-size inputs, the model is sensitive to the order in which input elements are presented. Precisely, different input permutations may provide different output probabilities because of the recurrent nature of the model, which causes the encoder hidden memory states to be devoid of information about the subsequent ones. As a result, the attention operation computes output probabilities with more informative encodings of the later processed input elements, making different decisions based on which input agent is missing information.

Regarding the training methodology, *Pointer Networks* can be trained using either a supervised or a reinforcement learning approach. In supervised learning, the model learns to execute decisions that lead to satisfactory solutions by approximating expert decisions. The optimal model parameters θ^*, i.e. the ones constituting a model indiscernible from the expert, are found maximizing of a cross-entropy loss

$$\theta^* = \arg\max_{\theta} \sum_{S,y} \sum_{i,j} y_i^j \log(p_i^j(S;\theta)), \tag{3.4}$$

where the training pair is composed of a collective formation instance S and an example solution y provided by the expert, with $y_i^j = 1$ if the solution contains agent s_i at position j and 0 otherwise.

In contrast, reinforcement learning employs *Pointer Networks* as a policy inside a decision-making process. Unlike supervised learning, decisions in reinforcement learning are learned through experience via pure exploration of the space of solutions rather than mimicking the ones of an expert. Although *Pointer Networks* can be trained under a reinforcement learning approach, it still suffers from being sensitive to the ordering of the input sequence. For this reason, *attention-based* models are presented in the next section as a solution to such problems.

3.4 Attention-based Models

Attention-based models emerged as a solution to the problem of the long-term dependency of words in *Natural Language Processing* (Bahdanau, Cho, and Bengio, 2014). Specifically, traditional encoder-decoder *Recurrent Neural Networks* have problems capturing the correlations among words in long sequences. To solve this problem, the attention operation explicitly learns the correlations between elements in sequences and assists the decoder by emphasizing the input elements that contribute the most to the current decoding decision.

Vaswani et al. (2017) noted that recurrent operations were no longer necessary for machine translation and proposed the *transformer*, an architecture based solely on the attention operation. Their approach consists of an encoder mapping an input sequence of symbol representations $(x_1, x_2, \ldots, x_n)$ to a set of continuous representations $h = (h_1, h_2, \ldots, h_n)$ from which a decoder generates an output of symbols $(y_1, y_2, \ldots, y_n)$.

The main processing unit of the transformer, i.e., the *multi-head attention* operation, is a mechanism adopted from search engines functioning as a mapping from queries $\boldsymbol{q}_i$ and keys $\boldsymbol{k}_j$ to attention weights a_{ij}. The inputs (queries and keys) are represented by a set of d_q and d_k dimensional vectors. The outputs (attention weights) are scalar values representing the normalized compatibility of the query $\boldsymbol{q}_i$ with the key $\boldsymbol{k}_j$. The first step in obtaining the attention weights is to compute the raw compatibilities

$$u_{ij} = \frac{(\boldsymbol{q}_i W^q)(\boldsymbol{k}_j W^k)^T}{\sqrt{d_k}}, \tag{3.5}$$

where W^q and W^k are two learnable linear transformations and the output is scaled by a factor of $\frac{1}{\sqrt{d_k}}$. Next, the attention weights are obtained by normalizing the compatibilities with a softmax function, i.e.,

$$a_{ij} = \frac{e^{u_{ij}}}{\sum_{j'} e^{u_{ij'}}}. \tag{3.6}$$

The transformer is composed of B attention blocks, through which hidden representations $\boldsymbol{h}_i^b$ ($b \in 1, 2, \ldots, B$) of different symbols i flow sequentially. Inside each block, attention weights are used to update the corresponding hidden representation $\boldsymbol{h}_i^b$ with contextual information, i.e., information about other hidden representations $\boldsymbol{h}_j^c$. To this end, attention weights are computed by considering $\boldsymbol{q}_i = \boldsymbol{h}_i^b$ and $\boldsymbol{k}_j = \boldsymbol{h}_j^c$, and a new hidden representation $\boldsymbol{h}_i^{b+1}$ is obtained by computing an attention weighted sum of contextual hidden representations $\boldsymbol{h}_i^c$,

$$\boldsymbol{h}_i^{b+1} = \sum_j a_{ij}(\boldsymbol{h}_j^c W^v), \tag{3.7}$$

where W^v is another learnable linear transformation.

Improving the approach presented above, *multi-head attention* distributes the computation of the attention weights over M different heads. To this end,

the keys are split into M chunks and processed in parallel by different attention operations. The hidden representations obtained from each head are summed together:

$$\boldsymbol{h}_i^{l+1} = \sum_m W_m^o \boldsymbol{h}_{im}^{l+1}, \tag{3.8}$$

where W^o is another learnable linear transformation. In practice, *multi-head attention* is often combined with other operations; traditionally, skip-connections, normalizing layers and fully-connected perceptron layers. Next, an *attention-based* approach conformed by the attention operation, and some additional operations will be presented for the purpose of generating collectives.

3.4.1 Generating Collectives

The generation of collectives with an *attention-based* model is formalized as a decision-making process where collectives are built incrementally by deciding which agents A are added to the collective S. More formally, the *attention-based* model receives the agents A represented by a list of d_a dimensional feature vectors, where d_a is the number of features. It is worth noting that, as is common in machine learning, the approach assumes that an agent can be represented as a vector of features, e.g., as the origin and destination locations in the ridesharing scenario or the personality traits and competence levels of students in the team formation scenario (see section 3.5 for more details). The model also receives a binary encoding of a collective $S = \{b_{1,S}, b_{2,S}, \ldots, b_{n,S}\}$, where $b_{i,S}$ are binary values determining whether the respective agents a_i are in the collective S or not. Therefore, the state of the problem at each step is represented by the tuple $s = (A, S)$.

Given a state s, the proposed *attention-based* is based on the encoder-decoder approach proposed by Kool, Van Hoof, and Welling (2018) for the *VRP*. The model performs as a stochastic policy $\pi_{\boldsymbol{\theta}}(s)$ parameterized by $\boldsymbol{\theta}$,

determining the probability for each agent in A to be included in the collective S. The encoder produces an embedding for each agent, i.e., a continuous representation of the input. Then, as illustrated in Figure 3.4, the decoder receives the embedding and the collective at each decoding step to compute the probabilities.

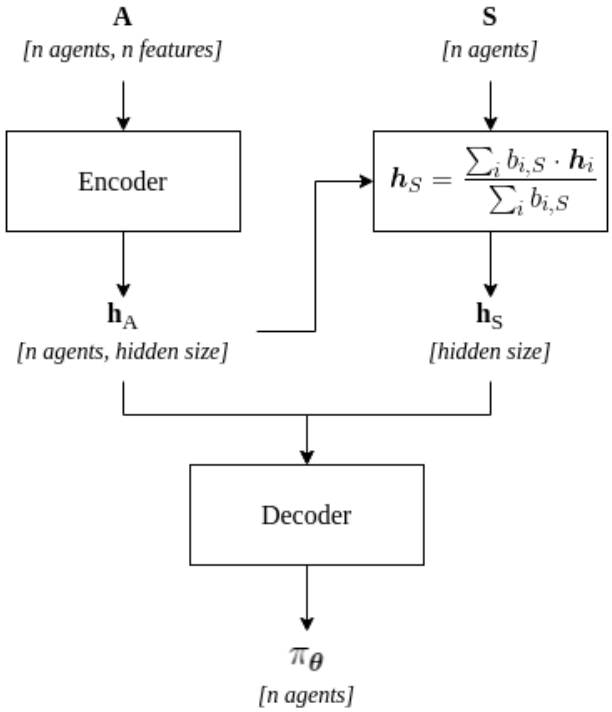


Figure 3.4: Encoder-decoder schema. The encoder produces a hidden embedding of the agents A and collective S. Then, given the embedding, the decoder computes the probability $\pi_{\boldsymbol{\theta}}$ for each agent in A to be added to the collective S. The size of the vector representation is specified for the input, output and intermediate hidden embeddings.

Encoder

The encoder is inspired by Vaswani et al. (2017)'s one, but unlike theirs, positional encoding is not required because the inputs in collective formation are order-invariant. Instead, an input feed-forward layer is employed to encode agents A from its d_A dimensional feature representation to a d_h dimensional embedding before the main attention blocks. In order to obtain the encoded

representation of the agents $\boldsymbol{h}_A$, the input embeddings are updated using N attention blocks[1], as depicted in Figure 3.5. Each block consists of two sub-layers: a multi-head attention layer and a feed-forward layer. Each sub-layer adds a residual connection (He et al., 2016) and performs layer normalization (Ba, Kiros, and Hinton, 2016) on its outputs, i.e., LayerNorm(x+sub-layer(x)). To facilitate residual connections, all sub-layers in the encoder use the same dimensionality d_h.

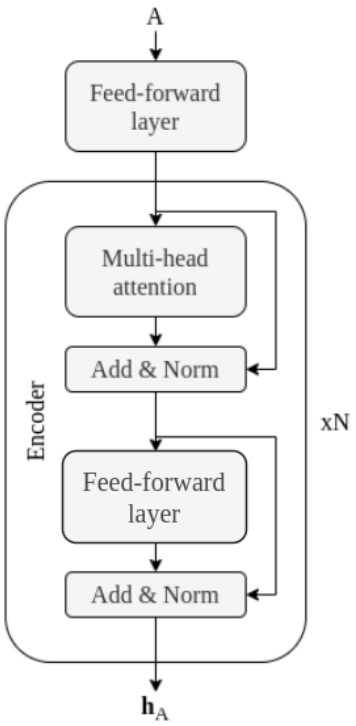


Figure 3.5: Encoder schema. The arrows represent the flow of data. The encoding process inside the white box is repeated N times.

Decoder

In order to compute the probabilities $\pi_{\boldsymbol{\theta}}(s)$, the decoder calculates the attention weights between the hidden representation of the agents $\boldsymbol{h}_A = \{\boldsymbol{h}_1, \boldsymbol{h}_2, \ldots, \boldsymbol{h}_n\}$

[1]In line with the notation from equation 3.7, $\boldsymbol{h}_A = \boldsymbol{h}_A^N$.

and a hidden representation of the collective $\boldsymbol{h}_S$. The hidden representation consists of a single dimension in order to obtain a scalar probability value for each agent. The following reduction is applied to the hidden representations of the agents $\boldsymbol{h}_A$ so as to produce a one-dimensional hidden representation of the collective $\boldsymbol{h}_S$:

$$\boldsymbol{h}_S = \frac{\sum_i b_{i,S} \cdot \boldsymbol{h}_i}{\sum_i b_{i,S}}. \tag{3.9}$$

At the initial state, the collective is empty, meaning that $\sum_i b_{i,S} = 0$. In that case, a d_h dimensional zero vector is used as a placeholder, $\boldsymbol{h}_S = \mathbf{0}$.

With the required hidden representations, $\boldsymbol{h}_A$ and $\boldsymbol{h}_S$, the decoder performs two last attention steps to obtain the probabilities $\pi_{\boldsymbol{\theta}}(s)$. Employing equation 3.7, the first attention step updates $\boldsymbol{h}_A = \boldsymbol{h}_A^N$ with contextual information about the collective previously encoded in $\boldsymbol{h}_S$. The second attention step computes the compatibilities u_{ij} between $\boldsymbol{h}_S$ and $\boldsymbol{h}_A^{N+1}$ with

$$u_i = \frac{(\boldsymbol{h}_S W^q)(\boldsymbol{h}_i^{N+1} W^k)^T}{d_h}. \tag{3.10}$$

Then, the compatibilities are normalized by applying a softmax to obtain the probability of each agent being added to a collective S,

$$\pi_{\boldsymbol{\theta},i}(s) = \frac{e^{\gamma \tanh u_i}}{\sum_j e^{\gamma \tanh u_j}}, \tag{3.11}$$

where $\gamma \tanh u_i$ controls the exploration of the model by shrinking the compatibilities to the range $[-\gamma, \gamma]$.

Learning with Maximum-Entropy Policy Gradient

Previously, an *attention-based* machine learning model has been presented as policy $\pi_{\boldsymbol{\theta}}(s)$ for the formation of collectives. This section presents maximum-entropy policy gradient as a training setup to optimize the model parameters

$\boldsymbol{\theta}$ for this task.

In order to optimize the model for collective formation, it is essential to recall the ultimate goal: forming a set of collectives $\mathcal{R}(A)$ from which a good-quality solution to the collective formation problem can be obtained by formulating an *ILP*. So as for the approach to be practical, it has to be guaranteed i) that $\mathcal{R}(A)$ contains collectives associated with high utility values by the cost function of the domain f, but also that ii) such a set contains a sufficient number of diverse candidate collectives. Such diversity is fundamental because, due to the non-overlapping constraint in (2.8), the optimal solution is likely to contain not only collectives with the highest possible value but also collectives of lower value. Henceforth, providing a sufficient number of alternatives to the *ILP* solver is crucial to achieving a final solution of good quality. To this end, the model is trained to optimize

$$\mathcal{L}(\theta|s) = \underbrace{\mathbb{E}_{\pi_{\boldsymbol{\theta}}(S|s)}\left[f(S)\right]}_{\text{Quality}} + \underbrace{\tau \cdot \mathcal{H}(\pi_{\boldsymbol{\theta}}(s))}_{\text{Diversity}}, \tag{3.12}$$

where $\mathcal{H}(\pi_{\boldsymbol{\theta}}(s))$ is the entropy of the model at state s, and τ weights the contribution of the entropy to the loss function. Optimizing the first term in (3.12) produces a policy that builds high utility collectives. In addition, the second term is introduced into the loss function to foster diversity.

Similar to Kool, Van Hoof, and Welling (2018), the model is optimized by gradient descent employing the well-known REINFORCE algorithm (Williams, 1992). However, in contrast to their work, the gradient considers the additional entropy term:

$$\nabla_{\boldsymbol{\theta}}\mathcal{L} = \mathbb{E}_{\pi_{\boldsymbol{\theta}}(S|s)}\left[\left(f(S) - b(s)\right)\nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(S|s)\right] + \tau\nabla_{\boldsymbol{\theta}}\mathcal{H}(\pi_{\boldsymbol{\theta}}(s)), \tag{3.13}$$

where $b(s)$ is a baseline to reduce the variance of $f(S)$. One popular choice to

get the baseline is using an exponential moving average of the observed characteristic function values. Another option is training a separate model, known as a critic, to predict the value of the characteristic function given a state s. While the first option does not provide a baseline for a particular state s, the second involves a complex training setup with two networks being optimized simultaneously, similar to *GANs*. Therefore, similar to Kool, Van Hoof, and Welling (2018), the characteristic function value is estimated employing a rollout baseline. Given a state s, a rollout uses the best policy obtained so far to generate a collective and reports its value.

Algorithm 1 REINFORCE with Rollout Baseline

Input: number of epochs E, number of iterations per epoch I, batch size B, significance α

Output: trained model parameters $\boldsymbol{\theta}$

Init $\boldsymbol{\theta}$
for $epoch = 1, \dots, E$ **do**
  for $iter = 1, \dots, I$ **do**
    $s_i \leftarrow \text{randomState}() \quad \forall i \in \{1, \dots, B\}$
    $S_i \leftarrow \text{rollout}(s_i, \boldsymbol{\theta}) \quad \forall i \in \{1, \dots, B\}$
    $S_i^{BL} \leftarrow \text{rollout}(s_i, \boldsymbol{\theta}_{BL}) \quad \forall i \in \{1, \dots, B\}$
    $\nabla \mathcal{L} \leftarrow \sum_{i=1}^{B} \left(f(S_i) - f(S_i^{BL}) \right) \nabla_{\boldsymbol{\theta}} \log \pi_{\boldsymbol{\theta}}(S_i | s_i)$
    $\nabla \mathcal{L}_{\mathcal{H}} \leftarrow \tau \sum_{i=1}^{B} \nabla_{\boldsymbol{\theta}} \mathcal{H}(\pi_{\boldsymbol{\theta}}(s_i))$
    $\boldsymbol{\theta} \leftarrow \text{Adam}(\boldsymbol{\theta}, \nabla \mathcal{L} + \nabla \mathcal{L}_{\mathcal{H}})$
  end for
  if $\text{OneSidedPairedTTest}(\pi_{\boldsymbol{\theta}}, \pi_{\boldsymbol{\theta}_{BL}}) \leq \alpha$ **then**
    $\boldsymbol{\theta}_{BL} \leftarrow \boldsymbol{\theta}$
  end if
end for
return $\boldsymbol{\theta}$

After computing the gradients, the model parameters $\boldsymbol{\theta}$ are updated using an Adam optimizer (Kingma and Ba, 2014). After completing one training epoch, the model is evaluated against the baseline by comparing their generated collectives with a paired t-test. If the current model outperforms the baseline, the baseline is updated with the model parameters. The entire train-

ing procedure is detailed in Algorithm 1.

Soft Baseline Update

Another popular method to update the baseline parameters, adopted by several reinforcement learning approaches (Lillicrap et al., 2015; Haarnoja et al., 2018), is to compute a moving average towards the model parameters after each iteration

$$\boldsymbol{\theta}_{BL} \leftarrow \lambda \cdot \boldsymbol{\theta} + (1 - \lambda) \cdot \boldsymbol{\theta}_{BL}, \tag{3.14}$$

where λ is the step size in the direction of the model parameters. This method softly updates the baseline parameters compared to the paired t-test approach, which updates the baseline only when the model outperforms it. The soft update produces a more robust training setup, usually leading to better performance of the trained models.

This update is integrated into algorithm 1 at the end of the inner *for loop*, replacing the t-test performed in line 11, which is omitted. The next chapter presents an empirical analysis conducted on the *attention-based* model, including a comparison of the different baseline updates. In addition, some experimental results concerning *GANs* and *Pointer Networks* will be presented to support the claim that they are inefficient approaches for collective formation.

3.5 Experimental Evaluation

This section provides an empirical examination of the general approach to collective formation introduced in the previous sections. The first section introduces the application domains and their corresponding state-of-the-art approaches, which serve as a baseline to confront the proposed approaches. Then, the empirical methodology is presented in detail. Finally, the last part

of this section provides empirical results on the solution quality achieved with the previously introduced approaches for the generation of collectives, i.e., *GANs*, *Pointer Networks* and *attention-based* models. A runtime analysis is not directly performed since the solution quality is evaluated on suboptimal approaches with the same time budget. The empirical results also include a detailed study of the *attention-based* model in terms of its ability to generate diverse collectives by quantifying the impact of the entropy term.

3.5.1 Application Domains

The primary goal of the experimental evaluation is to assess the performance of the general approach for collective formation in two structurally different real-world scenarios. On the one hand, the ridesharing scenario examined by Bistaffa et al. (2019) is considered, where the authors demonstrate that an algorithm strongly characterized by a greedy nature can produce solutions close to the optimal for hundreds of agents within one minute. On the other hand, the team formation scenario presented by Andrejczuk et al. (2019) is considered, which does not admit a greedy solution approach due to domain-specific constraints. Both of these domains are discussed in more detail in the following sections.

Ridesharing

The ridesharing scenario described in Bistaffa et al. (2019) takes place in a map of zones $\mathcal{Z} = \{z_1, z_2, \ldots, z_m\}$. An instance of this problem involves a pool of agents $A = \{a_1, a_2, \ldots, a_n\}$, where each agent wants to travel from an origin to a destination, formally $a_i \in \mathcal{Z} \times \mathcal{Z}$. This thesis considers collectives with cardinality $1 \leq |S| \leq 5$ to reflect the standard capacity of cars. Each

collective is assigned a value by a utility function $f(S)$:

$$f(S) = k \cdot E(S) + (1 - k) \cdot Q(S), \tag{3.15}$$

where $E_(S)$ quantifies the environmental benefits of forming the collective S, $Q(S)$ quantifies the quality of service incurred by the members of S, and $k \in [0, 1]$ controls the importance of each of the above-mentioned components. In the experimental evaluation, $k = 0.5$ is chosen to assess equal importance to environmental benefits and quality of service. The reader is referred to Bistaffa et al. (2019) for more details about each term in the utility function.

Team Formation

The team formation problem discussed in Andrejczuk et al. (2019) consists of a set of students $A = \{a_1, a_2, \ldots, a_n\}$, which aim at solving a task cooperatively in teams of equivalent performance. Andrejczuk et al. (2019) study four different tasks: Body rhythm, entrepreneur, art design and English. To simplify the experimental evaluation, this thesis focuses on the "English" task. Each student is represented by a tuple $(g, \boldsymbol{p}, \boldsymbol{l})$, where g is a binary value indicating the gender, $\boldsymbol{p}$ is a personality vector with four dimensions: sensing-intuition, thinking-feeling, extroversion-introversion, perception-judgement, each one evaluated in the range $[-1, 1]$. $\boldsymbol{l}$ is a vector measuring seven competence levels in the range $[0, 1]$: linguistic, logic-mathematics, visual-spatial, bodily-kinesthetic, musical, intrapersonal and interpersonal. For each task, at least one student in the team needs to cover different competencies. A utility function $f(S)$, proposed by Andrejczuk et al. (2019), assigns a value to each team based on how well it matches the requirements to carry out a specific task:

$$f(S) = \lambda \cdot u_{prof}(S, \tau) + (1 - \lambda) \cdot u_{con}(S), \tag{3.16}$$

where $u_{prof}(S, \tau)$ measures the proficiency of a team S on a task τ, $u_{con}(S)$ measures the congeniality of a team, and $\lambda \in [0, 1]$ controls the relative importance between this two terms. For more details on how each term is computed, the reader is referred to Andrejczuk et al. (2019). Because the goal of the team formation scenario proposed in Andrejczuk et al. (2019) is to obtain a balanced set of teams in order to foster cooperation and inclusiveness, the authors originally defined the corresponding collective formation problem as the maximization of a *Nash product*. Thus, to obtain a linear optimization problem, the problem is formulated as the sum of the logarithms of the teams' utility values. Here, the same linearized formalization is adopted.

3.5.2 Baselines

To assess the performance of the proposed approach in each of the domains, the state-of-the-art approaches proposed by Bistaffa et al. (2019) and Andrejczuk et al. (2019), respectively denoted as PG^2 and SynTeam, are employed. For both approaches, the parameters specified by the authors are used. Notice that, as mentioned in Section 2.2.3, these approaches already achieve near-optimal performance in their respective domains; hence the goal of this thesis is *not* to claim an improvement over these domain-specific solutions. Also, neither PG^2 nor SynTeam can be used outside of the domain in which they were originally designed. Thus, the goal is to demonstrate that the proposed general approach can achieve comparable performance to these approaches without being restricted to any specific application domain.

A comparison to state-of-the-art complete coalition structure generation approaches discussed in Section 2.2.1 is not contemplated since they cannot handle the number of agents considered in the following experiments.

Additionally, the proposed approach is confronted with the *MCTS* algorithm presented in Wu and Ramchurn (2020), which is the most recent general

approach for the formation of collectives. According to Wu and Ramchurn (2020), their approach employs a greedy rollout policy based on selecting collectives with the best value increment at each step. As previously mentioned, such a greedy policy can not be directly applied in the team formation domain, preventing the approach in Wu and Ramchurn (2020) from finding any feasible solution in this case.

For this reason, the experiments contemplate a second version of *MCTS* employing a heuristic preventing the choice of actions leading to a potentially unfeasible collective during rollout. This heuristic uses symmetries in the search space to avoid branches of the tree which lead to a permutation of a previously explored collective. In order to provide a complete set of experiments, these also contemplate a standard version of *MCTS* employing a random rollout, i.e., one selecting actions from a uniform distribution. These three *MCTS* approaches are referred to as *G-MCTS* (greedy), *A-MCTS* (i.e., adapted) and *R-MCTS* (i.e., random), respectively.

3.5.3 Generative Adversarial Networks

This section presents the results of the experimental evaluation conducted to assess the performance of *GANs* as a model to generate candidate collectives for the general approach.

Empirical Methodology

To assess the capacity of *GANs* to generate a set of good candidate collectives, a comparison of the candidates generated by *GANs* and PG^2 has been directly conducted in the ridesharing domain. For the comparison, each model receives an instance and employs a time budget of 40 seconds to generate as many collectives. Then, the process is repeated for 20 ridesharing instances, and the

values of the generated collectives are aggregated in the form of histograms to be able to be compared in shape and size.

Training - Training times may vary depending on the size of the training instances, but in general, it takes between 12 and 24 hours on the employed machine (2.20 GHz CPU, 128 GB RAM and NVIDIA RTX 2080 Ti GPU).

Hyperparameters - The *GAN* is implemented in PyTorch. The model parameters are initialized with $Uniform(-1/\sqrt{d},\ 1/\sqrt{d})$, where d is the input size. The generator is composed of 4 feedforward hidden layers, including batch normalization and a *ReLU* activation function. An output feedforward layer with only a *sigmoid* activation is employed. The input sizes of each layer are 32 (latent vector size), 128, 256, 512 and 1024, respectively. The output size of the last layer is set to the size of the ridesharing solutions, which is fixed at 20. The discriminator is composed of 3 feedforward hidden layers, including a *ReLU* activation function. An output feedforward layer with no activation function is employed. The input sizes of each layer are 20 (solution size), 512, 256 and 128, respectively. As is usual for the *GAN* discriminator, the output size of the last layer is 1 since it outputs a single scalar value. The model is trained during 100 epochs consisting of 400 batches with 256 instances each. Concerning the optimization of the model parameters, a learning rate of 10^{-4} is employed.

Solution Quality

GANs, similar to supervised approaches, require thousands of examples used during training to show the model what structure high-value solutions to the collective formation share. Thus, the model is trained on a dataset obtained by sampling random collectives and selecting the ones over a specific threshold. Increasing the threshold might improve solution quality but also increase the time required to generate a sufficient amount of collectives to train the

model. For the experiments, a threshold of $f(\mathbf{S}) \geq 1$ has been employed in the ridesharing domain as it is fast to obtain collectives of decent performance. The choice of the threshold depends on the characteristic function of the domain. As a result, the threshold must be tuned based on prior knowledge of the distribution of values, which represents a limitation since previous domain knowledge is not available in general. Figure 3.6 shows the distribution of values of collectives in the dataset obtained for the ridesharing domain, which needs to be learned by the *GAN*.

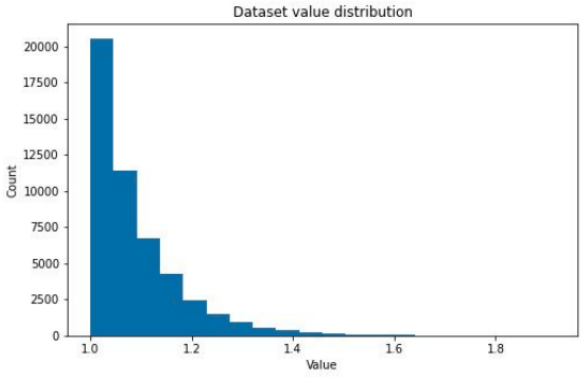
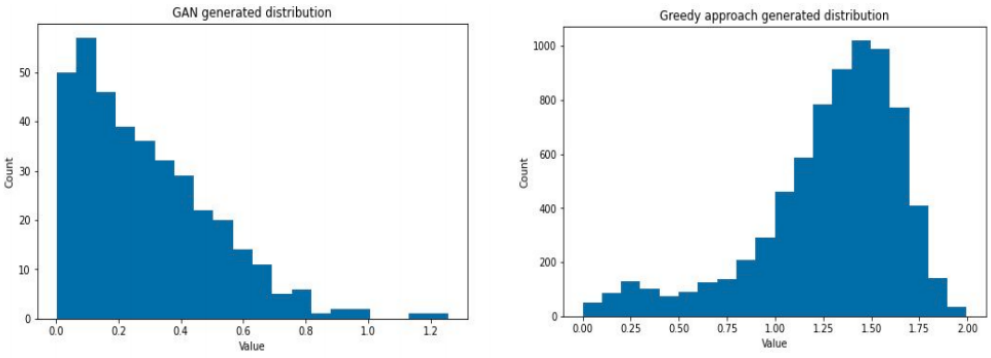


Figure 3.6: Distribution of values of the collectives in the training set of the ridesharing domain.

Although the *GAN* learns to generate collectives better than sampling them randomly, 3.7 shows that very few values satisfy $f(\mathbf{S}) \geq 1$, while most of them are under this threshold. Therefore, the *GAN* does not correctly replicate the distribution of values present in the dataset. In addition, comparing it to the greedy approach by Bistaffa et al. (2019), the quality of the collectives is far from being useful for the proposed general approach.



(a) GAN (b) PG^2

Figure 3.7: Comparison of the distribution of values of collectives generated for ridesharing by the GAN and the PG^2.

In order to improve the performance, one could choose to increase the threshold, so the *GAN* learns from higher quality collectives. This potentially presents two problems. First, the dataset generation procedure could be costly in terms of computation time. Second, increasing the threshold too much could remove some collectives, which, despite being of lower value, they form close-to-optimal solutions because they synergize with other high-value collectives. In any case, because the *GAN* does not manage to capture the dataset distribution adequately, other promising approaches (*pointer networks* and *attention-based models*) better encoding collectives as variable-size sequences are considered in the following experiments.

3.5.4 Pointer Networks

This section presents the results of the experimental evaluation conducted to assess the performance of *pointer networks* as a model to generate candidate collectives for the general approach.

Empirical Methodology

To assess the capacity of *pointer networks* to generate a set of good candidate collectives, a comparison of the performance of the general approach is evaluated when employing *pointer networks* and PG^2 for generating collectives in the ridesharing domain. A total of 50 problem instances are considered for ridesharing. For each instance, 40 seconds are employed to generate collectives and 20 to solve the *ILP* with the generated collectives. Then, the optimality ratio is computed for each solution, i.e., the ratio between the average of the obtained solution values and the value of the optimal solution, which is obtained by solving (2.8) to optimality.

Training - Training times may vary depending on the size of the training instances, but in general, it takes between 12 and 24 hours on the employed machine (2.20 GHz CPU, 128 GB RAM and NVIDIA RTX 2080 Ti GPU).

Hyperparameters - The *pointer network* is implemented in PyTorch. The model parameters are initialized with $Uniform(-1/\sqrt{d}, 1/\sqrt{d})$, where d is the input size. The encoder is composed of an embedding feedforward layer and a *recurrent neural network*, both with a hidden size of 512. The decoder is composed of the attention operation previously described, with a single head and a hidden size of 512. The model is trained during 100 epochs consisting of 200 batches with 256 instances each. Concerning the optimization of the model parameters, a learning rate of 10^{-4} is employed.

Solution Quality

Despite generating a large dataset of high-value collectives by random sampling ensuring following a domain agnostic data generation procedure, it is impractical for the purpose of the empirical analysis. Therefore, since an approach generating high-value collectives faster exists (PG^2), the *pointer net-*

work model is trained under a supervised approach employing the example solutions provided by PG^2, which follows the value distribution in figure 3.7 (b).

The evaluation is conducted on different enhancements introduced to the standard *pointer network* originally proposed by Vinyals, Fortunato, and Jaitly (2015). The first introduces bidirectional *RNNs* to address the issue that a token in the input sequence is only informed with previous tokens (Schuster and Paliwal, 1997). The second one employs *teacher forcing* (Williams and Zipser, 1989), a technique used to avoid accumulating errors during training by using the ground truth as input in the decoding step. Table 3.1 shows the average optimality ratio obtained with the different *pointer network* variants and the baseline for ridesharing *PG²*.

Model	Ratio
PG^2	0.98
Pointer Network	0.48
+ Bidirectional	0.47
+ Teacher Forcing	0.69

Table 3.1: Pointer Network results.

The best performing *pointer network* variant is the one incorporating *teacher forcing*, suggesting that the error introduced by inaccuracies carried during sequential decisions has a significant negative impact on the training process. Nevertheless, the *pointer network* is far from producing collectives with an optimality ratio similar to the one achieved by the ad-hoc approach *PG²*. Adding to this the fact that *pointer networks* are not order-invariant, *attention-based* models are evaluated as an improved approach to overcoming this limitation. Moreover, different from previous approaches, *attention-based* models are trained under a *reinforcement learning* setup to avoid being tied to a pre-existing solution for generating high-quality collectives.

3.5.5 Attention-based Model

This section presents the results of the experimental evaluation conducted to assess the performance of *attention-based* model to generate candidate collectives for the general approach.

Empirical Methodology

For each case study, the algorithms mentioned above are evaluated using real-world datasets considered by Bistaffa et al. (2019) and Andrejczuk et al. (2019). A total of 50 problem instances are considered for ridesharing, and 20 problem instances are considered for team formation. For each instance, each algorithm finds a solution using 50 different seeds (i.e., $0, \ldots, 49$). Then, the optimality ratio is computed for each solution, i.e., the ratio between the average of the obtained solution values and the value of the optimal solution, which is obtained by solving (2.8) to optimality. Finally, the average over all instances of such optimality ratios is confronted between the different approaches. Standard deviations are not reported since, in all experiments, they are less than 0.02.

Training & Evaluation - The runtime and the hardware employed for training and evaluation are the following:

- Training times may vary depending on the size of the training instances, but in general, it takes between 12 and 24 hours on the employed machine (2.20 GHz CPU, 128 GB RAM and NVIDIA RTX 2080 Ti GPU).
- For evaluation, a total time budget of 60 seconds is considered. The portion of the time budget devoted to each part of the proposed approach is determined using IRACE (López-Ibáñez et al., 2016), a widely used software for tuning algorithmic parameters. The optimal portion of the time budget devoted to the generation of promising candidates is 50

seconds, during which the *attention-based* model can generate tens of thousands of collectives.

The *attention-based* model is implemented in PyTorch and the *ILP* solver employed is *CPLEX 20.1.0.0*.

Hyperparameters -The model parameters are initialized with $Uniform(-1/\sqrt{d}, 1/\sqrt{d})$, where d is the input size. The attention operation consists of 8 heads and a hidden layer size of $d = 256$, whereas the feed-forward layers use $d = 512$. The encoder is composed of 3 attention blocks. The models are trained during 100 epochs consisting of 400 batches with 256 instances each. Regarding evaluation, 100 batches of data are considered with the same number of instances each. Concerning the optimization of the model parameters, a learning rate of 10^{-4} is employed, and a significance of $\alpha = 0.05$ for the one-sided paired t-test is considered.

Solution Quality

This section compares the proposed approach employing the *attention-based* model for the generation of collectives and the baselines discussed in Section 4.2.1 on the two considered collective formation domains. Table 3.2 reports the results of the experiments on the ridesharing domain. The best-performing *attention-based* model is comparable with PG^2 for $n = 50$, but PG^2 still outperforms this model for larger sizes. This result is not surprising since PG^2 has been specifically designed for this domain.

Results also show that the optimality ratio obtained with the *attention-based* model is superior compared to the *MCTS* approaches (including *R-MCTS*, the only *MCTS* approach in the comparison that does not incorporate any domain-specific subroutine), which cannot compute a solution of acceptable quality. Moreover, in any experiments for $n = 200$, the *G-MCTS* approach

n	AM (t-test)	AM (soft)	*G-MCTS*	*A-MCTS*	*R-MCTS*	PG^2
50	0.89	0.96	0.92	0.09	0.00	0.98
100	0.76	0.90	0.88	0.04	0.04	0.98
200*	0.65	0.87	–	0.01	0.02	1.00

Table 3.2: Optimality ratio for the *attention-based* model (AM) and for baseline approaches in the ridesharing domain. For all experiments, a time budget of 60 seconds was used. Missing values ("–") indicate that the approach did not compute any solution better than the initial one within the time budget. *For $n = 200$, the ratio with respect to the PG^2 solution value is measured since computing the optimal in a manageable amount of time is not possible.

by Wu and Ramchurn (2020) could not compute a solution better than the initial one (i.e., all singletons with a total utility of 0) in the considered time budget. A possible explanation could be that the utility function considered involves more computation than the ones studied in (Wu and Ramchurn, 2020), where utilities are directly obtained by sampling from a uniform distribution. These results could suggest that complex utility functions might hinder the performance of *MCTS* in large-scale scenarios.

As for the update rule on the baseline, the approach implementing the soft update outperforms the one implementing the t-test update.

Table 3.3 reports the results of the experiments on the team formation domain. As for ridesharing, the optimality ratio obtained in that case is significantly better than the one obtained by other *MCTS* approaches, even the one adapted explicitly for this domain (i.e., *A-MCTS*). Moreover, compared to SynTeam, it can be observed that the gap between the proposed approach and the domain-specific approach for team formation is smaller than for ridesharing, especially for the larger problem instances.

In contrast to the ridesharing domain, there is no evident difference between the t-test update and the soft update for the baseline for team formation. The most reasonable explanation is that the optimality ratio is already quite good compared to the one in the ridesharing domain; hence the impact of soft-update

n	AM (t-test)	AM (soft)	*G-MCTS*	*A-MCTS*	*R-MCTS*	ST
50	0.97	0.97	–	0.84	–	0.99
60	0.95	0.93	–	0.78	–	0.99
100*	0.92	0.91	–	–	–	1.00

Table 3.3: Optimality ratio for the *attention-based* model (AM) and for baseline approaches in the team formation domain. ST indicates the results for the SynTeam approach. For all experiments, a time budget of 60 seconds was used. Missing values ("–") indicate that the approach did not compute any solution better than the initial one within the time budget. *For $n = 100$, the ratio with respect to the solution computed by SynTeam is reported since computing the optimal in a manageable amount of time is not possible.

in team formation is less pronounced.

Impact of Entropy

One of the most important components of the *attention-based* model is the entropy term in Equation 3.13, which allows the model to generate a wider variety of candidates, hence providing more options to the *ILP* solver and resulting in a better overall performance of the approach. In this section, the impact of entropy is evaluated concerning the variety of candidates generated for the ridesharing and team formation domains. To this end, a second set of experiments is conducted, comparing the distribution over the values of the collectives generated by the model with and without entropy ($\tau = 0.05$ and $\tau = 0.00$, respectively)[2] to the distribution over the values of the collectives in the optimal solution. All these distributions are obtained by performing a Gaussian Kernel Density Estimation (Silverman, 2018) over thousands of collectives generated by the models, given a problem instance.

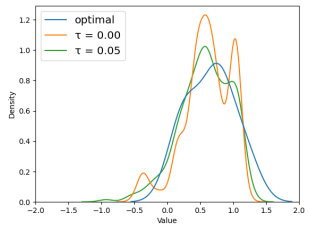
Figures 3.8 and 3.9 report the distribution over the values for six different example instances of the ridesharing and team formation domains. By looking at the distribution of the values, it can be observed that the model with higher

[2]Increasing the τ parameter over 0.05 affected the convergence of the model negatively during training.

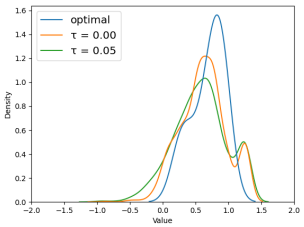
entropy covers a broader range of values than the one with lower entropy. In this respect, it can be observed that the model with higher entropy produces distributions closer to the optimal ones when compared with the distribution produced by the model with lower entropy. Indeed, a model will perform better when it generates a distribution of values similar to the ones in the optimal solution.

In order to measure such a similarity, the *Kullback-Leibler* divergence is measured between the distribution produced by the models (standard and entropy-enhanced) and the distribution of values in the optimal solution. The distribution of values obtained employing the entropy-enhanced model is closer to the optimal one in 157 instance over a total of 200 in the ridesharing domain. Moreover, the average divergence over these samples is lower for the model with higher entropy, and the p-value is $3 \cdot 10^{-6}$. The analysis in the team formation domain further confirms these results. In this case, the entropy-enhanced model produces a distribution closer to the optimal one for all test instances.

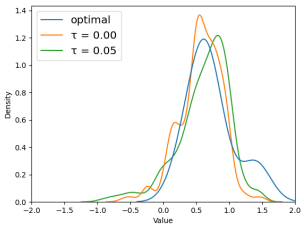
Overall, these results confirm that adding entropy in the process of generating collectives is beneficial for collective formation problems since the model associated with a higher entropy can generate collectives with lower values that, when combined with higher-valued ones, lead to better solutions computed by the *ILP*.



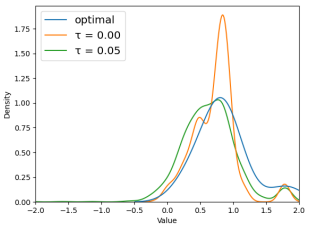
(a) First instance



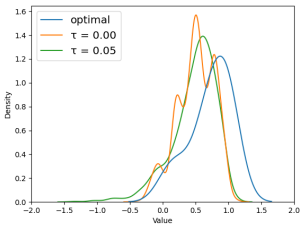
(b) Second instance



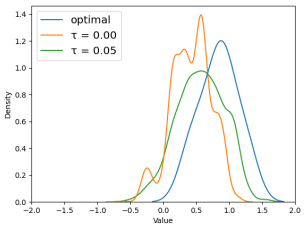
(c) Third instance



(d) Fourth instance

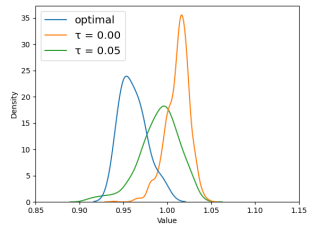


(e) Fifth instance

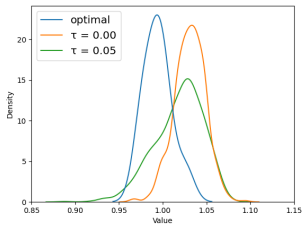


(f) Sixth instance

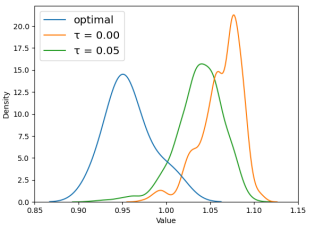
Figure 3.8: Probability density of the values of collectives generated by the *attention-based* model employing $\tau = 0.00$ and $\tau = 0.05$ for different ridesharing instances.



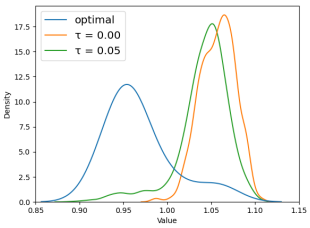
(a) First instance



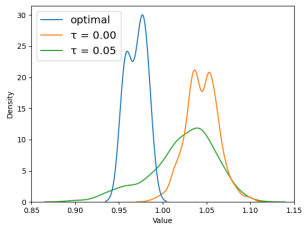
(b) Second instance



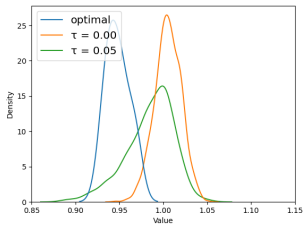
(c) Third instance



(d) Fourth instance



(e) Fifth instance



(f) Sixth instance

Figure 3.9: Probability density of the values of collectives generated by the *attention-based* model employing $\tau = 0.00$ and $\tau = 0.05$ for different team formation instances.

Chapter 4

Capacitated Multi-Agent Pick-up and Delivery

The previous chapter presented a general approach that, by incorporating into a classical approach a machine learning component learning the domain structure, achieves to provide solutions of high quality to a variety of collective formation problems. However, the formation of collectives is sometimes entangled with other optimization goals. For instance, this chapter studies the capacitated *MAPD* problem, which combines the assignment of pickup and delivery tasks (formulated as a collective formation problem) with the planning of the paths and resolution of the derived conflicts. Thus, the task assignment component needs to account for the paths and conflicts in order to produce an assignment leading to optimal paths.

The following sections present the approach for capacitated *MAPD*, introducing minor changes to the *attention-based* model in order to be used for the assignment of pickup and delivery tasks to the agents. The resulting approach combines the *attention-based* model, which performs the task assignment, with *CBS*, which implements a conflict resolution strategy to plan conflict-free paths. An empirical evaluation is conducted to assess the per-

formance of the proposed approach by comparing it to various approaches, including some state-of-the-art. Finally, a discussion on how to overcome the limitations of the *attention-based* model for the capacitated *MAPD* problem is provided at the end of the chapter.

4.1 Solution Approach

This section proposes a solution approach for capacitated *MAPD* introducing an *attention-based* model to assign tasks to agents. Following the approach by Chen et al. (2021), the task assignment and path planning are integrated, such that the task assignment component can decide based on the increment of path cost derived from conflicts occurring during the planning of paths of specific agents. Their approach prioritizes assigning tasks to agents by considering the total travel distance experienced by an agent incorporating a specific task into its route. Despite working well in practice, the decisions performed by their approach are intrinsically greedy, affecting the final solution cost. Instead, the *attention-based* task assignment proposed in this thesis learns by observing the total travel cost of the final assignment, going beyond greedy decisions.

On a high level, the *attention-based* task assignment and the approach by Chen et al. (2021) construct an assignment incrementally, following the instructions provided in algorithm 2. Initially, the algorithm receives the set of agents A and the set of tasks T of a particular problem instance. Then, the approach, starting with an empty assignment S_i for each agent a_i, keeps adding tasks to the assignments S_i based on a priority heap H until all tasks are assigned. The priority heap H consists of elements $H_{ij} \in \mathbb{R}$ indicating the priority of assigning task j to agent i. The difference between both approaches resides in how they compute the priority heap, i.e. how they model function $h(\mathcal{S}, A, T)$. In Chen et al. (2021), the authors propose to use a minimum

cost assignment based on the total travel delay. Instead, this thesis proposes to learn the priority heap leading to the best expected final total travel cost with an *attention-based* model. Another important difference between the two approaches is how they perform path-planning. The computation of the paths in Chen et al. (2021) is implicit inside function h. Because it integrates path planning, the assignment can decide by considering the paths. However, its decisions only account for the agents which already have planned their path, thus leading to potentially suboptimal assignments. In contrast, the *attention-based* task assignment, performing as h, does not consider the paths of the agents. Instead, the paths are planned optimally when the assignment of tasks has finished. Still, a disadvantage derived from not integrating path-planning is that the *attention-based* model is not provided with the exact paths but only with the pickup and delivery locations.

Algorithm 2 Integrated approach for task assignment

Input: A set of agents A, a set of tasks T

$\mathcal{S} \leftarrow \{S_1, S_2, \ldots, S_{|A|}\}$ where $S_i \leftarrow \emptyset \quad \forall a_i \in A$
while $T \neq \emptyset$ **do**
  $H \leftarrow h(\mathcal{S}, A, T)$
  $i', j' \leftarrow argmin(H)$
  $S_{i'} \leftarrow S_{i'} \cup t_{j'}$
  $T = T - t_{j'}$
end while
return $\mathcal{S}$

The *attention-based* model employed here to compute the priority heap H is very similar to the one used for general collective formation problems in the previous chapter. Nevertheless, in this case, a collective of tasks must be formed based on the initial position of agents A. The next section presents the structure of the *attention-based* model designed to consider this new problem dimension.

4.1.1 Attention-based Task Assignment

In order to perform the task assignment, the *attention-based* model needs to compute the priority heap values H_{ij}, corresponding to the priority of assigning task j to agent i. To this end, the agents A (represented by a vector with the agents starting position) and tasks T (represented by a vector with the pickup and delivery locations for each task) are encoded to obtain their respective hidden representations $\boldsymbol{h}_A$ and $\boldsymbol{h}_T$ by two independent encoders, both adopting the same architecture as the one presented in figure 3.5. The hidden representation of the collective $\boldsymbol{h}_S$ is obtained by computing the average of the hidden representation of tasks present in the collective, as in equation 3.9. Then, because a collective of tasks needs to incorporate also the agent that is executing them, the hidden representation of the collective $\boldsymbol{h}_S$ is updated by concatenating the hidden dimensions of $\boldsymbol{h}_A$ and $\boldsymbol{h}_S$, and passing the output through a feed-forward (FF) layer to obtain a hidden representation of the same size:

$$\boldsymbol{h}_S = FF(concat(\boldsymbol{h}_S, \boldsymbol{h}_A)). \tag{4.1}$$

Then, once $\boldsymbol{h}_S$ has been updated, the compatibilities (equation 3.10) between $\boldsymbol{h}_S$ and $\boldsymbol{h}_T$ are computed to calculate the output probabilities with equation 3.11.

Apart from requiring an encoding for agents and tasks, the *attention-based* model for the task assignment is significantly different from the model for collective formation in that the tasks and collectives encodings are performed for each agent in order to obtain a vector indicating the probability of assigning the tasks to each agent. As a result, the encoder-decoder *attention-based* model for task assignment (figure 4.1) adds a new dimension to the variables it considers, in contrast to the encoder-decoder *attention-based* model for collective formation (figure 3.4). In the *attention-based* model for task assignment, there

is one output probability vector for each agent i indicating the probability of selecting each task j, which values define the probability heap, i.e., $H_{ij} = \pi_{\boldsymbol{\theta} ij}$.

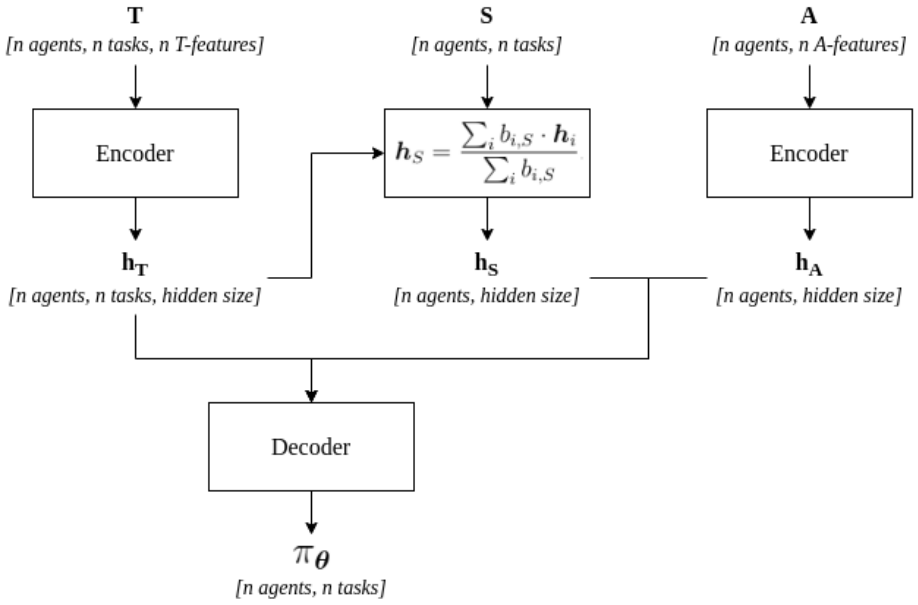


Figure 4.1: Encoder-decoder schema. The encoder produces a hidden embedding of the tasks T, agents A and collective S. Then, the decoder computes the probabilities $\pi_{\boldsymbol{\theta}}$ for each task in T to be added to the collective S corresponding to agent A. The size of the vector representation is specified for the input, output and intermediate hidden embeddings.

Learning Optimal Assignments with Policy Gradient

The previous section describes how to adapt the *attention-based* model for collective formation to perform the assignment of tasks to agents in a capacitated *MAPD* problem. This section presents the training setup to learn optimal assignments for this problem.

In order to obtain an optimal assignment, the model needs to learn the assignment producing the minimum total travel distance travelled by the agents while carrying the assigned tasks. To this end, the model, defining a parametrized policy $\pi_{\boldsymbol{\theta}}(\mathcal{S}|s)$, is trained to optimize

$$\mathcal{L}(\theta|s) = \mathbb{E}_{\pi_{\boldsymbol{\theta}}(\mathcal{S}|s)}\left[f(\mathcal{S})\right], \tag{4.2}$$

where $f(\mathcal{S})$ is the total travel distance of the assignment $\mathcal{S}$.

As in the previous chapter, the model is optimized by gradient descent with the REINFORCE algorithm, described in algorithm 1. However, the entropy term is not included in equation 4.2 because diversity is no longer a requirement as there is no need to aggregate single agent assignments by means of an *ILP*. Concerning the baseline update rule, the soft baseline update is employed as it is superior to the *t-test* update, as demonstrated in the previous chapter.

The next section presents the experimental evaluation comparing the performance of the *attention-based* model to other existing assignment approaches, including the integrated *RMCA* approach, when employed for assigning pickup and delivery tasks for the capacitated *MAPD* problem.

4.2 Experimental Evaluation

In this section, the *attention-based* model is evaluated as a task assignment approach for the capacitated *MAPD* problem. The following parts of this section describe the baselines confronted with the *attention-based* model and the methodology employed to train and evaluate the model. Finally, the solution quality is presented in terms of the total travel distance of the agents and the number of conflicts derived from an assignment.

4.2.1 Baselines

Three baseline approaches are considered to assess the performance of the *attention-based* model for the task assignment in the capacitated *MAPD* problem. The first one, denoted as *Hungarian*, employs the Hungarian method to recursively assign tasks to agents based on the total travel distance computed without considering conflicts. Thus, this approach has two sources of error; first, its decisions are guided by a greedy component since they only account

for the immediate impact on the total travel distance instead of considering the effect of possible future decisions, and second, it is agnostic about conflicts. The second considered approach, denoted as *TSP*, consists in formulating the assignment problem as a *TSP* and solving it employing the *OR-Tools* software (Perron and Furnon, 2022). Although the greedy component is no longer present in this approach, the path planning component is still agnostic about conflicts since it only views the pickup and delivery locations given as a distance matrix whose values correspond to the total travel distance, not considering the conflicts. The third and last approach is the one by Chen et al. (2021) integrating task assignment and path planning, which incorporates conflicts into the assignment. Still, the assignment is performed recursively, similar to the *Hungarian* approach, thus also suffering from greedy decisions.

4.2.2 Empirical Methodology

To assess the capacity of the *attention-based* model to produce task assignments leading to an optimal total travel distance, the model has been compared to the *Hungarian*, the *TSP* and the *RMCA* approaches. First, each approach (except *RMCA*) is employed to perform a task assignment over 50 instances, each consisting of 20 agents with a fixed capacity of 3 and 50 tasks. Then, the resulting assignments are passed to *PBS* (Ma et al., 2019), a variant of the *CBS* algorithm with prioritized conflict resolution, and the total travel distance and the number of conflicts at the root of *PBS* are reported. The *RMCA* approach employs the same instances and, by performing task assignment and conflict resolution simultaneously, reports the total travel distance. The number of root conflicts produced by the *RMCA* approach is computed by aggregating the conflicts encountered during execution but only once for each agent in order to make a fair comparison with other approaches which compute the number of root conflicts and do not consider conflicts in the rest

of the branch. All the approaches use a total time budget of 600 seconds for the whole process, despite computation times varying a lot from $\sim$ 10 seconds to $\sim$ 300 seconds between different instances. Only a few instances exceeded the time budget, in which case they were removed from the analysis.

Environment - The experiments are performed on a 21×35 4-neighbour grid, representing the structure of a warehouse, depicted in figure 4.2. This grid structure, proposed by Ma et al., 2017, guarantees that a solution to a problem instance exists as long as the agents and task pickup and delivery locations are initialized in the so-called task endpoints (dark green cells in figure 4.2).

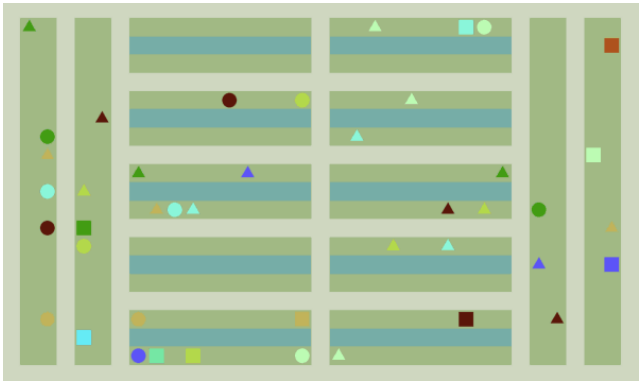


Figure 4.2: 21×35 4-neighbour grid representing the structure of the warehouse where the empirical analysis of the capacitated *MAPD* problem has been conducted. Dark cells are blocked. Dark green cells are task endpoints. Squares, circles and triangles are agents, pickup and delivery locations respectively.

Training - Training times may vary depending on the size of the training instances, but in general, it takes between 12 and 24 hours on the employed machine (2.20 GHz CPU, 128 GB RAM and NVIDIA RTX 2080 Ti GPU).

Hyperparameters - The *attention-based* is implemented in PyTorch. The model parameters are initialized with $Uniform(-1/\sqrt{d},\ 1/\sqrt{d})$, where d is the input size. The attention operation consists of 8 heads and a hidden layer size of $d = 256$, whereas the feed-forward layers use $d = 512$. The encoder is composed of 3 attention blocks. The models are trained during 100 epochs

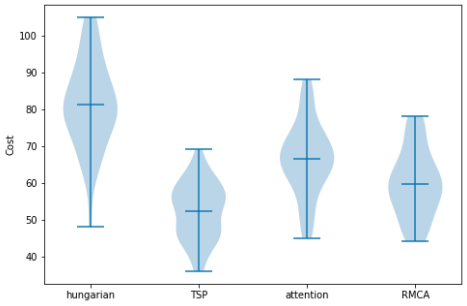
consisting of 400 batches with 256 instances each. Regarding evaluation, 100 batches of data are considered with the same number of instances each. Concerning the optimization of the model parameters, a learning rate of 10^{-4} is employed, and a significance of $\alpha = 0.05$ for the one-sided paired t-test is considered.

4.2.3 Experimental Results

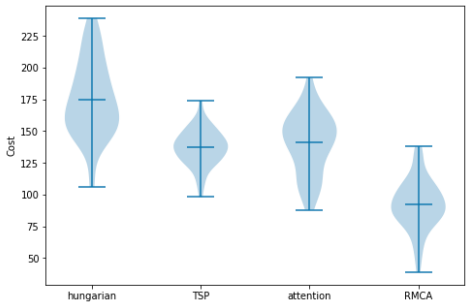
The results of the experimental evaluation are presented in figure 4.3. In that figure, the solution cost shows the total travel distance distribution over the evaluated instances for the four studied approaches. The number of root conflicts is presented in the same figure with the purpose of obtaining an indicator of how well each approach takes into account the possible conflicts derived from the task assignment.

On smaller instances (10 agents and 20 tasks), *TSP* is the best-performing approach. The explanation is that *TSP* produces an optimal solution in the absence of conflicts, and small instances tend to have fewer conflicts. In the second place, the *attention-based* model and *RMCA* show similar performance. Moreover, all the approaches show a similar amount of conflicts at the root of *CBS*, confirming that the number of conflicts in small instances does not impact the solution quality.

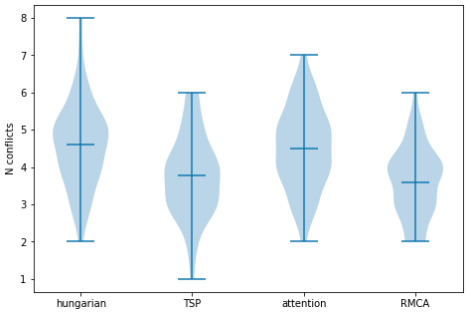
On larger instances (20 agents and 50 tasks), while the integrated *RMCA* approach is superior in all aspects, the attention model is significantly better than the *Hungarian* approach and shows similar performance as the *TSP* approach. However, by looking at the number of root conflicts in *CBS*, the *attention-based* model seems unable to reduce them. Thus, it is likely that the *attention-based* model decides on the assignment only guided by the locations of pickup and delivery points, not by the potential conflicts. Nonetheless, considering both aspects is essential for achieving solutions closer to the optimal



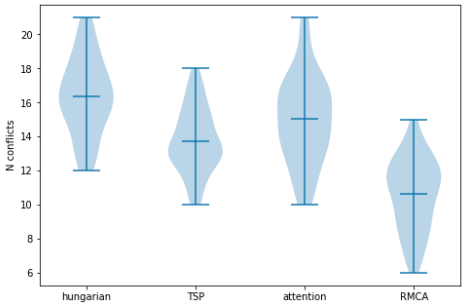
(a) Solution cost for 10 agents and 20 tasks.



(b) Solution cost for 20 agents and 50 tasks.



(c) Root conflicts for 10 agents and 20 tasks.



(d) Root conflicts for 20 agents and 50 tasks.

Figure 4.3: Comparison of the solution cost and the number of conflicts at the root of *CBS* between four task assignment approaches: *Hungarian* method, *TSP* formulation, *attention-based* model and the integrated *RMCA* approach.

one. Therefore, the following section discusses possible solutions to overcome this limitation.

4.3 Limitations of the *Attention-based* Task Assignment

In the previous sections, the *attention-based* model has been compared with other task assignment approaches for the capacitated *MAPD* problem. Results showed the *attention-based* model is not able to capture the knowledge related to the conflicts and cannot take decisions by considering these. Although one could argue that, in practice, the *attention-based* model could potentially

learn to infer a mapping from the pickup and delivery locations assigned to an agent to its path in order to provide an assignment minimizing the total travel distance, the experimental results show that this exceeds the learning capabilities of the model, as the number of conflicts does not seem to be reduced. Therefore, this section discusses two different directions in order to make the approach knowledgeable about the impact of conflicts in the paths of the agents.

The first direction focuses on enhancing the machine learning component, i.e., the *attention-based* model, with knowledge about the paths. More precisely, by encoding a representation of the paths $\mathcal{P}$ into a hidden representation $h_{\mathcal{P}}$, and combining it into the representation of a collective as in equation 4.1. The representation of the paths is obtained by a path-planning approach and can incorporate different levels of knowledge:

1. Optimal ordered sequence of pickup and delivery locations minimizing the travel distance without considering conflicts (obtained solving the *TSP* formulation).

2. Optimal paths at the root of *CSB* without considering conflicts.

3. Optimal paths considering conflicts, obtained by fully solving the path-planning and conflict resolution with *CBS*.

These different representations are ordered from the one which requires less computation but also introduces less knowledge to the one requiring more computation but also introduces more knowledge about the paths. In all these alternatives, the paths are represented by an ordered sequence. Thus, the encoder must recognize the order in the input sequence. An interesting solution to this challenge is incorporating the positional encoding employed in Vaswani et al. (2017).

The second direction accepts that the machine learning component lacks the required abilities to reason about possible constraints derived from the paths and proposes an anytime improvement strategy that starts with the assignment provided by the *attention-based* model and, by computing the travel distance of each assignment by means of *CBS*, it replaces the assignments with the highest travel distance with other assignments also proposed by the *attention-based* model. With this solution, the *attention-based* model would need to be trained considering the entropy term to generate a diverse of possible assignments.

Chapter 5

Conclusions and Future Work

This thesis has focused on collective formation, a challenging problem present in many real-world applications connected to Sustainable Development Goals. Existing solution approaches for the formation of collectives rely on domain-specific instructions introduced by an expert with prior knowledge about the structure of the application domain. The lack of generality is the main issue of existing approaches, which are devised for specific applications. This thesis proposes a novel approach introducing machine learning techniques for problems involving the formation of collectives. The proposed approach consists of a combination of a machine learning component and an *ILP* solver. The machine learning model is trained to generate high-quality candidate collectives by learning the structure of a particular application domain. The *ILP* solver computes a solution to the collective formation problem by finding the optimal set of non-overlapping collectives over the set of high-quality candidate collectives produced by the machine learning component. The proposed approach is able to compete with the state-of-the-art approaches for collective formation in terms of solution quality while not requiring specific ad-hoc instructions introduced to enhance the performance of the approaches on the specific domains. Therefore, the proposed approach can be used for poten-

tially new collective formation problems without requiring ad-hoc algorithmic development. In particular, this thesis showed that:

1. The quality of the solutions provided by the proposed approach can compete with the quality of the solutions provided by the PG^2 and *SynTeam* approaches on the ridesharing and team formation domains. The proposed approach also outperforms the most general collective formation approach based on *MCTS* on both domains.

2. Unlike PG^2 and *SynTeam*, the machine learning model can learn the structure of high-quality collectives in the ridesharing and team formation domains without the need for further algorithm design.

3. Among the different machine learning models evaluated (i.e., *GANs*, *pointer networks* and the *attention-based* model), the *attention-based* model better adapts to the collective formation task because it allows variable-size inputs and outputs, also guaranteeing order invariance. This is confirmed by the experimental evaluation, which shows that the *attention-based* model outperforms *GANs* and *Pointer networks*.

4. Concerning the training methodology, since reinforcement learning does not require examples to learn from, it is a better framework for building general approaches than supervised learning.

5. A fundamental aspect that contributes to improving the quality of the solutions produced by the proposed approach has been the introduction of the entropy term in the objective function in order to enhance diversity in the set of promising collectives generated by the *attention-based* model. Diversity is critical to provide the *ILP* solver with more varied candidates from which to construct better solutions.

6. Extending the use of the *attention-based* model to other approaches involving the formation of collectives requires significant changes in the architecture. Although the task assignment problem only requires an additional encoder, when combined with conflict resolution and path planning in the capacitated *MAPD* problem, the assignment generated by the *attention-based* model does not account for the derived conflicts.

Overall, this thesis offers a general-purpose approach that, without specialized domain knowledge, can be used to solve a variety of real-world collective formation problems involving hundreds of agents. The analysis and discussion conducted in this thesis of the various methodologies set the stage towards the application of the approach in other combinatorial optimization problems involving the formation of collectives.

Despite the above-mentioned contribution to the state-of-the-art, the work presented in this thesis can be improved and extended according to several interesting future research lines. First, a promising direction is the study of the proposed general approach to other collective formation problems, e.g. in the coordination of robots in emergency scenarios (Ramchurn et al., 2010) or in collective energy purchasing (Vinyals et al., 2012). Second, the study of methodologies to incorporate external constraints into the proposed approach also represent an interesting future research line. For instance, the *attention-based* approach is not able to produce collectives which satisfy requirements such as the student's preferences in team formation or car availability in ridesharing. Last, and probably the most challenging direction, is extending the use of the *attention-based* model to problems which combine collective formation with other combinatorial problems. For instance, with the study of the capacitated *MAPD* problem, this thesis aims to reveal the limitations of the current model in such scenarios and what improvements can contribute to extending the use of the model to more complex domains.

Bibliography

[1] Brandon Amos and J Zico Kolter. "Optnet: Differentiable optimization as a layer in neural networks". In: *International Conference on Machine Learning*. PMLR. 2017, pp. 136–145.

[2] Ewa Andrejczuk, Filippo Bistaffa, Christian Blum, Juan A Rodríguez-Aguilar, and Carles Sierra. "Synergistic team composition: A computational approach to foster diversity in teams". In: *Knowledge-Based Systems* 182 (2019), p. 104799.

[3] Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. "Layer normalization". In: *arXiv preprint arXiv:1607.06450* (2016).

[4] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473* (2014).

[5] Christopher A Bailey, Timothy W McLain, and Randal W Beard. "Fuel-saving strategies for dual spacecraft interferometry missions". In: *The Journal of the astronautical sciences* 49.3 (2001), pp. 469–488.

[6] Max Barer, Guni Sharon, Roni Stern, and Ariel Felner. "Suboptimal variants of the conflict-based search algorithm for the multi-agent pathfinding problem". In: *Seventh Annual Symposium on Combinatorial Search*. 2014.

[7] Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. "Neural combinatorial optimization with reinforcement learning". In: *arXiv preprint arXiv:1611.09940* (2016).

[8] Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. "Machine learning for combinatorial optimization: a methodological tour d'horizon". In: *European Journal of Operational Research* 290.2 (2021), pp. 405–421.

[9] Bernd Bischl, Pascal Kerschke, Lars Kotthoff, Marius Lindauer, Yuri Malitsky, Alexandre Fréchette, Holger Hoos, Frank Hutter, Kevin Leyton-Brown, Kevin Tierney, et al. "Aslib: A benchmark library for algorithm selection". In: *Artificial Intelligence* 237 (2016), pp. 41–58.

[10] Filippo Bistaffa, Christian Blum, Jesús Cerquides, Alessandro Farinelli, and Juan A Rodríguez-Aguilar. "A computational approach to quantify the benefits of ridesharing for policy makers and travellers". In: *IEEE Transactions on Intelligent Transportation Systems* 22.1 (2019), pp. 119–130.

[11] Filippo Bistaffa, Georgios Chalkiadakis, and Alessandro Farinelli. "Efficient Coalition Structure Generation via Approximately Equivalent Induced Subgraph Games". In: *IEEE Transactions on Cybernetics* (2021).

[12] Christian Blum and Andrea Roli. "Metaheuristics in combinatorial optimization: Overview and conceptual comparison". In: *ACM computing surveys (CSUR)* 35.3 (2003), pp. 268–308.

[13] Eli Boyarski, Ariel Felner, Roni Stern, Guni Sharon, David Tolpin, Oded Betzalel, and Eyal Shimony. "ICBS: Improved conflict-based search algorithm for multi-agent pathfinding". In: *Twenty-Fourth International Joint Conference on Artificial Intelligence*. 2015.

[14] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. "Language models are few-shot learners". In: *Advances in neural information processing systems* 33 (2020), pp. 1877–1901.

[15] Jesus Cerquides, Alessandro Farinelli, Pedro Meseguer, and Sarvapali D Ramchurn. "A tutorial on optimization for multi-agent systems". In: *The Computer Journal* 57.6 (2014), pp. 799–824.

[16] Georgios Chalkiadakis, Edith Elkind, and Michael Wooldridge. "Computational aspects of cooperative game theory". In: *Synthesis Lectures on Artificial Intelligence and Machine Learning* 5.6 (2011), pp. 1–168.

[17] Donald Chan and Daniel Mercier. "IC insertion: an application of the travelling salesman problem". In: *The International Journal of Production Research* 27.10 (1989), pp. 1837–1841.

[18] Minas Chatzos, Ferdinando Fioretto, Terrence WK Mak, and Pascal Van Hentenryck. "High-fidelity machine learning approximations of large-scale optimal power flow". In: *arXiv preprint arXiv:2006.16356* (2020).

[19] Zhe Chen, Javier Alonso-Mora, Xiaoshan Bai, Daniel D Harabor, and Peter J Stuckey. "Integrated task assignment and path planning for capacitated multi-agent pickup and delivery". In: *IEEE Robotics and Automation Letters* 6.3 (2021), pp. 5816–5823.

[20] Nicos Christofides. *Worst-case analysis of a new heuristic for the travelling salesman problem.* Tech. rep. Carnegie-Mellon Univ Pittsburgh Pa Management Sciences Research Group, 1976.

[21] IBM ILOG Cplex. "V12. 1: User's Manual for CPLEX". In: *International Business Machines Corporation* 46.53 (2009), p. 157.

[22] Mehul Damani, Zhiyao Luo, Emerson Wenzel, and Guillaume Sartoretti. "PRIMAL _2: Pathfinding via reinforcement and imitation multi-agent learning-lifelong". In: *IEEE Robotics and Automation Letters* 6.2 (2021), pp. 2666–2673.

[23] Viet Dung Dang and Nicholas R Jennings. "Generating coalition structures with finite bound from the optimal guarantees". In: (2004).

[24] George B Dantzig and John H Ramser. "The truck dispatching problem". In: *Management science* 6.1 (1959), pp. 80–91.

[25] Xiaotie Deng and Christos H Papadimitriou. "On the complexity of cooperative solution concepts". In: *Mathematics of operations research* 19.2 (1994), pp. 257–266.

[26] Jian-Ya Ding, Chao Zhang, Lei Shen, Shengyin Li, Bing Wang, Yinghui Xu, and Le Song. "Accelerating primal solution findings for mixed integer programs based on solution prediction". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 02. 2020, pp. 1452–1459.

[27] Priya Donti, Brandon Amos, and J Zico Kolter. "Task-based end-to-end model learning in stochastic optimization". In: *Advances in neural information processing systems* 30 (2017).

[28] Marco Dorigo and Luca Maria Gambardella. "Ant colonies for the travelling salesman problem". In: *biosystems* 43.2 (1997), pp. 73–81.

[29] Adam N Elmachtoub and Paul Grigas. "Smart "predict, then optimize"". In: *Management Science* 68.1 (2022), pp. 9–26.

[30] Alessandro Farinelli, Manuele Bicego, Filippo Bistaffa, and Sarvapali D Ramchurn. "A hierarchical clustering approach to large-scale near-optimal coalition formation with quality guarantees". In: *Engineering Applications of Artificial Intelligence* 59 (2017), pp. 170–185.

[31] Ferdinando Fioretto, Terrence WK Mak, and Pascal Van Hentenryck. "Predicting ac optimal power flows: Combining deep learning and lagrangian dual methods". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 34. 01. 2020, pp. 630–637.

[32] Ferdinando Fioretto, Pascal Van Hentenryck, Terrence WK Mak, Cuong Tran, Federico Baldo, and Michele Lombardi. "Lagrangian duality for constrained deep learning". In: *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*. Springer. 2021, pp. 118–135.

[33] Maxime Gasse, Didier Chételat, Nicola Ferroni, Laurent Charlin, and Andrea Lodi. "Exact combinatorial optimization with graph convolutional neural networks". In: *Advances in Neural Information Processing Systems* 32 (2019).

[34] Ralph E Gomory. "Outline of an algorithm for integer solutions to linear programs and an algorithm for the mixed integer problem". In: *50 Years of Integer Programming 1958-2008*. Springer, 2010, pp. 77–103.

[35] Ian J Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron C Courville, and Yoshua Bengio. "Generative Adversarial Nets". In: *NIPS*. 2014.

[36] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, and Sergey Levine. "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor". In: *International conference on machine learning*. PMLR. 2018, pp. 1861–1870.

[37] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[38] John H Holland. "Genetic algorithms and adaptation". In: *Adaptive control of ill-defined systems*. Springer, 1984, pp. 317–333.

[39] John J Hopfield and David W Tank. ""Neural" computation of decisions in optimization problems". In: *Biological cybernetics* 52.3 (1985), pp. 141–152.

[40] André Hottung and Kevin Tierney. "Neural large neighborhood search for the capacitated vehicle routing problem". In: *arXiv preprint arXiv:1911.09539* (2019).

[41] Taoan Huang, Bistra Dilkina, and Sven Koenig. "Learning Node-Selection Strategies in Bounded Suboptimal Conflict-Based Search for Multi-Agent Path Finding". In: *International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2021.

[42] Taoan Huang, Sven Koenig, and Bistra Dilkina. "Learning to resolve conflicts for multi-agent path finding with conflict-based search". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 13. 2021, pp. 11246–11253.

[43] Taoan Huang, Jiaoyang Li, Sven Koenig, and Bistra Dilkina. "Anytime Multi-Agent Path Finding via Machine Learning-Guided Large Neighborhood Search". In: (2022).

[44] Samuel Ieong and Yoav Shoham. "Marginal contribution nets: A compact representation scheme for coalitional games". In: *Proceedings of the 6th ACM Conference on Electronic Commerce*. 2005, pp. 193–202.

[45] JQ James, Wen Yu, and Jiatao Gu. "Online vehicle routing with neural combinatorial optimization and deep reinforcement learning". In: *IEEE Transactions on Intelligent Transportation Systems* 20.10 (2019), pp. 3806–3817.

[46] Chaitanya K Joshi, Thomas Laurent, and Xavier Bresson. "An efficient graph convolutional network technique for the travelling salesman problem". In: *arXiv preprint arXiv:1906.01227* (2019).

[47] Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. "Learning combinatorial optimization algorithms over graphs". In: *Advances in neural information processing systems* 30 (2017).

[48] Elias Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. "Learning to branch in mixed integer programming". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 30. 1. 2016.

[49] Philip Kilby, John Slaney, Toby Walsh, et al. "The backbone of the travelling salesperson". In: *IJCAI*. Citeseer. 2005, pp. 175–180.

[50] Diederik P Kingma and Jimmy Ba. "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980* (2014).

[51] Wouter Kool, Herke Van Hoof, and Max Welling. "Attention, learn to solve routing problems!" In: *arXiv preprint arXiv:1803.08475* (2018).

[52] James Kotary, Ferdinando Fioretto, and Pascal Van Hentenryck. "Learning hard optimization problems: A data generation perspective". In: *Advances in Neural Information Processing Systems* 34 (2021), pp. 24981–24992.

[53] James Kotary, Ferdinando Fioretto, Pascal Van Hentenryck, and Bryan Wilder. "End-to-end constrained optimization learning: A survey". In: *arXiv preprint arXiv:2103.16378* (2021).

[54] Markus Kruber, Marco E Lübbecke, and Axel Parmentier. "Learning when to use a decomposition". In: *International conference on AI and*

OR techniques in constraint programming for combinatorial optimization problems. Springer. 2017, pp. 202–210.

[55] Bert FJ La Maire and Valeri M Mladenov. "Comparison of neural networks for solving the travelling salesman problem". In: *11th Symposium on Neural Network Applications in Electrical Engineering*. IEEE. 2012, pp. 21–24.

[56] Ailsa H Land and Alison G Doig. "An automatic method for solving discrete programming problems". In: *50 Years of Integer Programming 1958-2008*. Springer, 2010, pp. 105–132.

[57] Jan Karel Lenstra and David Shmoys. *The Traveling Salesman Problem: A Computational Study*. 2009.

[58] Zhuwen Li, Qifeng Chen, and Vladlen Koltun. "Combinatorial optimization with graph convolutional networks and guided tree search". In: *Advances in neural information processing systems* 31 (2018).

[59] Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver, and Daan Wierstra. "Continuous control with deep reinforcement learning". In: *arXiv preprint arXiv:1509.02971* (2015).

[60] Minghua Liu, Hang Ma, Jiaoyang Li, and Sven Koenig. "Task and path planning for multi-agent pickup and delivery". In: *Proceedings of the International Joint Conference on Autonomous Agents and Multiagent Systems (AAMAS)*. 2019.

[61] Andrea Lodi. "Mixed integer programming computation". In: *50 years of integer programming 1958-2008*. Springer, 2010, pp. 619–645.

[62] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Mauro Birattari, and Thomas Stützle. "The irace package: Iterated racing for automatic algorithm configuration". In: *Operations Research Perspectives* 3 (2016), pp. 43–58.

[63] Hang Ma, Daniel Harabor, Peter J Stuckey, Jiaoyang Li, and Sven Koenig. "Searching with consistent prioritization for multi-agent path finding". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 33. 01. 2019, pp. 7643–7650.

[64] Hang Ma, Jiaoyang Li, TK Kumar, and Sven Koenig. "Lifelong multi-agent path finding for online pickup and delivery tasks". In: *arXiv preprint arXiv:1705.10868* (2017).

[65] Tomasz Michalak, Talal Rahwan, Edith Elkind, Michael Wooldridge, and Nicholas R Jennings. "A hybrid exact algorithm for complete set partitioning". In: *Artificial Intelligence* 230 (2016), pp. 14–50.

[66] Olof Mogren. "C-RNN-GAN: Continuous recurrent neural networks with adversarial training". In: *arXiv preprint arXiv:1611.09904* (2016).

[67] Roger B Myerson. "Graphs and cooperation in games". In: *Mathematics of operations research* 2.3 (1977), pp. 225–229.

[68] Vinod Nair, Sergey Bartunov, Felix Gimeno, Ingrid von Glehn, Pawel Lichocki, Ivan Lobov, Brendan O'Donoghue, Nicolas Sonnerat, Christian Tjandraatmadja, Pengming Wang, et al. "Solving mixed integer programs using neural networks". In: *arXiv preprint arXiv:2012.13349* (2020).

[69] Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takác. "Reinforcement learning for solving the vehicle routing problem". In: *Advances in neural information processing systems* 31 (2018).

[70] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization: algorithms and complexity*. Courier Corporation, 1998.

[71] Fernando Peres and Mauro Castelli. "Combinatorial optimization problems and metaheuristics: Review, challenges, design, and development". In: *Applied Sciences* 11.14 (2021), p. 6449.

[72] Laurent Perron and Vincent Furnon. *OR-Tools*. Version v9.5. Google, Nov. 25, 2022. URL: `https://developers.google.com/optimization/`.

[73] Pavel A Pevzner, Haixu Tang, and Michael S Waterman. "An Eulerian path approach to DNA fragment assembly". In: *Proceedings of the national academy of sciences* 98.17 (2001), pp. 9748–9753.

[74] Talal Rahwan and Nick Jennings. "Coalition structure generation: Dynamic programming meets anytime optimisation". In: (2008).

[75] Talal Rahwan, Tomasz P Michalak, and Nicholas R Jennings. "Minimum search to establish worst-case guarantees in coalition structure generation". In: *Twenty-Second International Joint Conference on Artificial Intelligence*. 2011.

[76] Talal Rahwan, Tomasz P Michalak, Michael Wooldridge, and Nicholas R Jennings. "Coalition structure generation: A survey". In: *Artificial Intelligence* 229 (2015), pp. 139–174.

[77] Talal Rahwan, Sarvapali D Ramchurn, Nicholas R Jennings, and Andrea Giovannucci. "An anytime algorithm for optimal coalition structure generation". In: *Journal of artificial intelligence research* 34 (2009), pp. 521–567.

[78] Sarvapali D Ramchurn, Mariya Polukarov, Alessandro Farinelli, Nick Jennings, and Cuong Trong. "Coalition formation with spatial and temporal constraints". In: (2010).

[79] Sebastian Raschka. "Model evaluation, model selection, and algorithm selection in machine learning". In: *arXiv preprint arXiv:1811.12808* (2018).

[80] Gabriele Röger and Malte Helmert. "The more, the merrier: Combining heuristic estimators for satisficing planning". In: *Twentieth International Conference on Automated Planning and Scheduling*. 2010.

[81] Sebastian Ruder. "An overview of gradient descent optimization algorithms". In: (2016).

[82] Tuomas Sandholm, Kate Larson, Martin Andersson, Onn Shehory, and Fernando Tohmé. "Coalition structure generation with worst case guarantees". In: *Artificial intelligence* 111.1-2 (1999), pp. 209–238.

[83] Guillaume Sartoretti, Justin Kerr, Yunfei Shi, Glenn Wagner, TK Satish Kumar, Sven Koenig, and Howie Choset. "Primal: Pathfinding via reinforcement and imitation multi-agent learning". In: *IEEE Robotics and Automation Letters* 4.3 (2019), pp. 2378–2385.

[84] Farah Sarwar and Abdul Aziz Bhatti. "Critical analysis of hopfield's neural network model and heuristic algorithm for shortest path computation for routing in computer networks". In: *Proceedings of 2012 9th International Bhurban Conference on Applied Sciences & Technology (IBCAST)*. IEEE. 2012, pp. 115–119.

[85] Mike Schuster and Kuldip K Paliwal. "Bidirectional recurrent neural networks". In: *IEEE transactions on Signal Processing* 45.11 (1997), pp. 2673–2681.

[86] Guni Sharon, Roni Stern, Ariel Felner, and Nathan R Sturtevant. "Conflict-based search for optimal multi-agent pathfinding". In: *Artificial Intelligence* 219 (2015), pp. 40–66.

[87] Onn Shehory and Sarit Kraus. "Methods for task allocation via agent coalition formation". In: *Artificial intelligence* 101.1-2 (1998), pp. 165–200.

[88] David Silver. "Cooperative pathfinding". In: *Proceedings of the aaai conference on artificial intelligence and interactive digital entertainment*. Vol. 1. 1. 2005, pp. 117–122.

[89] Bernard W Silverman. *Density estimation for statistics and data analysis*. Routledge, 2018.

[90] Yuan Sun, Xiaodong Li, and Andreas Ernst. "Using statistical measures and machine learning for graph reduction to solve maximum weight clique problems". In: *IEEE transactions on pattern analysis and machine intelligence* 43.5 (2019), pp. 1746–1760.

[91] Cuong Tran, Ferdinando Fioretto, and Pascal Van Hentenryck. "Differentially private and fair deep learning: A lagrangian dual approach". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 35. 11. 2021, pp. 9932–9939.

[92] Long Tran-Thanh, Tri-Dung Nguyen, Talal Rahwan, Alex Rogers, and Nicholas R Jennings. "An efficient vector-based representation for coalitional games". In: (2013).

[93] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[94] Meritxell Vinyals, Filippo Bistaffa, Alessandro Farinelli, and Alex Rogers. "Coalitional energy purchasing in the smart grid". In: *2012 IEEE International Energy Conference and Exhibition (ENERGYCON)*. IEEE. 2012, pp. 848–853.

[95] Oriol Vinyals, Meire Fortunato, and Navdeep Jaitly. "Pointer networks". In: *Advances in neural information processing systems* 28 (2015).

[96] Po-Wei Wang, Priya Donti, Bryan Wilder, and Zico Kolter. "Satnet: Bridging deep learning and logical reasoning using a differentiable satisfiability solver". In: *International Conference on Machine Learning*. PMLR. 2019, pp. 6545–6554.

[97] Yong Wang and Zunpu Han. "Ant colony optimization for traveling salesman problem based on parameters optimization". In: *Applied Soft Computing* 107 (2021), p. 107439.

[98] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. "A survey of transfer learning". In: *Journal of Big data* 3.1 (2016), pp. 1–40.

[99] Ronald J Williams. "Simple statistical gradient-following algorithms for connectionist reinforcement learning". In: *Machine learning* 8.3 (1992), pp. 229–256.

[100] Ronald J Williams and David Zipser. "A learning algorithm for continually running fully recurrent neural networks". In: *Neural computation* 1.2 (1989), pp. 270–280.

[101] Feng Wu and Sarvapali D Ramchurn. "Monte-Carlo tree search for scalable coalition formation". In: *Proceedings of the 29th International Joint Conference on Artificial Intelligence (IJCAI)*. 2020, pp. 407–413.

[102] Álinson S Xavier, Feng Qiu, and Shabbir Ahmed. "Learning to solve large-scale security-constrained unit commitment problems". In: *INFORMS Journal on Computing* 33.2 (2021), pp. 739–756.

[103] Tao Xu, Pengchuan Zhang, Qiuyuan Huang, Han Zhang, Zhe Gan, Xiaolei Huang, and Xiaodong He. "Attngan: Fine-grained text to image generation with attentional generative adversarial networks". In: *Pro-*

ceedings of the IEEE conference on computer vision and pattern recognition. 2018, pp. 1316–1324.

[104] D Yun Yeh. "A dynamic programming approach to the complete set partitioning problem". In: *BIT Numerical Mathematics* 26.4 (1986), pp. 467–474.

[105] Shuyang Zhang, Jiaoyang Li, Taoan Huang, Sven Koenig, and Bistra Dilkina. "Learning a Priority Ordering for Prioritized Planning in Multi-Agent Path Finding". In: *Proceedings of the International Symposium on Combinatorial Search*. Vol. 15. 1. 2022, pp. 208–216.

Sommario

Questa tesi considera il problema della formazione di collettivi di agenti per domini di applicazione che permettono di raggiungere gli *Obiettivi di Sviluppo Sostenibile* (ad esempio, mobilità condivisa e apprendimento cooperativo). Tali problemi richiedono approcci performancti che possano produrre soluzioni di alta qualità per centinaia di agenti. Con questo obiettivo in mente, le soluzioni esistenti per la formazione di collettivi si concentrano sullo sfruttare le caratteristiche specifiche del dominio di applicazione considerato. Tuttavia, gli approcci risultanti non sono trasferibili ad altri problemi di formazione di collettivi. Pertanto, è necessario studiare approcci generali che non richiedano conoscenza preliminare del dominio, tali da essere applicati a domini differenti.

Su questa linea, questa tesi propone un approccio generale per la formazione di collettivi basato su una innovativa combinazione tra *machine learning* e un *integer linear program*. Più precisamente, un componente di *machine learning* viene addestrato per generare un insieme di collettivi candidati che hanno la possibilità di fare parte di una soluzione. Quindi, tali collettivi e i loro valori di utilità corrispondenti vengono introdotti in un *integer linear program* che trova una soluzione al problema. In questo modo, la componente di machine learning apprende la struttura che caratterizza i collettivi più promettenti in un particolare dominio, rendendo l'intero approccio valido per diverse applicazioni.

Inoltre, l'analisi empirica condotta su due domini di applicazione realistici (mobilità condivisa e apprendimento cooperativo) dimostra che l'approccio proposto fornisce soluzioni di qualità paragonabile agli approcci allo stato dell'arte per ciascun dominio.

Infine, questa tesi mostra anche che l'approccio proposto può essere esteso a problemi che combinano la formazione di collettivi con altri obiettivi di ottimizzazione. Più precisamente, questa tesi propone un'estensione dell'approccio di formazione di collettivi per l'assegnazione di posizioni di ritiro e consegna agli robot in un magazzino. La valutazione sperimentale mostra che, sebbene sia possibile utilizzare l'approccio di formazione di collettivi per questo scopo, sono necessari diversi miglioramenti per competere con gli approcci più avanzati.

Nel complesso, questa tesi si pone l'obiettivo di dimostrare che il *machine learning* può essere combinato con successo con gli approcci classici di ottimizzazione per la formazione di collettivi, apprendendo la struttura di un dominio, riducendo la necessità di algoritmi concepiti ad-hoc per un dominio di applicazione specifico.