

University of the Basque Country (UPV/EHU)



Universidad del País Vasco Euskal Herriko Unibertsitatea

Department of Communications Engineering
Faculty of Engineering in Bilbao
NQaS | Networking, Quality and Security Research Group

Thesis submitted for the degree of Doctor of Philosophy

**Neural Combinatorial Optimization as an enabler
technology to design real-time Virtual Network
Function placement decision systems**

Ruben Solozabal

Supervisors: Dr. Fidel Liberal and Dr. Bego Blanco

October, 2020

Ruben Solozabal

Neural Combinatorial Optimization as an enabler technology to design real-time Virtual Network Function placement decision systems

Thesis submitted for the degree of Doctor of Philosophy, October, 2020

Supervisors: Dr. Fidel Liberal and Dr. Bego Blanco

University of the Basque Country (UPV/EHU)

NQaS | Networking, Quality and Security Research Group

Faculty of Engineering in Bilbao

Department of Communications Engineering

Plaza Ingeniero Torres Quevedo, 1

48013 - Bilbao

Resumen ejecutivo

El tráfico y el número de servicios ofrecidos a través de la red móvil están experimentando un aumento exponencial. La creciente demanda conlleva una mejora de recursos en infraestructura continua para seguir manteniendo una experiencia de usuario satisfactoria. Hasta el momento, esta demanda ha sido solventada evolucionando el marco tecnológico existente en la red móvil (3G, 4G), consiguiéndose una mejora en capacidad de red, cobertura, así como en eficiencia de la misma. Sin embargo, las propuestas de modelo 5G apuntan por un cambio de paradigma, la siguiente generación de red móvil debe además soportar servicios dispares tales como servicios de ultra-baja latencia, Internet of Things, aplicaciones de misión crítica, etc.

A fin de dar soporte a servicios con características tan distintas sobre una misma infraestructura, la red 5G se fundamenta sobre tecnologías de virtualización de red (Network Function Virtualization -NFV- y Software Defined Networks -SDN-). Esto representa un avance significativo en la industria de las telecomunicaciones. La virtualización de red otorga una flexibilidad sin precedentes a la hora de gestionar no solamente los recursos de red sino también los servicios que se ofertan sobre ella. En este sentido, los despliegues tradicionales realizados con elementos de red físicos 'hard-wired' dan paso funciones de red virtuales (VNF) capaces de ejecutarse sobre plataformas de cómputo general. Lo que facilita el despliegue y mantenimiento de los mismos, reduciendo los costes operativos asociados.

Además, la arquitectura NFV se encarga de monitorizar y garantizar el cumplimiento de los requerimientos de servicio. Para ello existe una interacción vertical entre los distintos estratos de la infraestructura con el objetivo de mantener la calidad de servicio acordado. En este sentido, la flexibilidad que otorga la red permite realizar tareas de optimización a fin de mejorar la eficiencia de la red. Optimizar los recursos de la infraestructura es vital para la viabilidad de esta solución. Sin embargo, la obtención de algoritmos capaces de adaptarse al estado de la red supone un reto tecnológico. En este sentido, uno de los aspectos más relevantes propuestos es la capacidad de otorgar inteligencia directamente a la red mediante la utilización de Machine Learning.

Objetivo

El objetivo de esta tesis es proponer mecanismos eficientes de despliegue de servicios sobre arquitecturas 5G mediante esquemas NFV. En particular, se han estudiado mecanismos de optimización en el ámbito del «VNF Placement». La optima colocación de recursos en la infraestructura es un problema de combinatoria discreta NP-hard en el que se busca una solución que optimice el despliegue dadas las restricciones vinculadas al tipo de servicio contratado y las características de la infraestructura. Concretamente, el objetivo perseguido es la minimización del consumo energético.

Existe una gran variedad de estudios que analizan algoritmos de «VNF Placement» que van desde métodos exactos (mathematical programming o constraint programming) los cuales suponen un alto coste computacional además de requerir un elevado tiempo de obtención de soluciones; a

algoritmos heurísticos y metaheurísticos utilizados para obtener aproximaciones al problema en tiempos razonables. Sin embargo, dadas las características particulares del entorno, es necesario particularizar el estudio y buscar soluciones capaces de adaptarse a los requerimientos de sistema. En este sentido se busca que la solución responda a las peticiones de servicio de forma interactiva, lo que obliga a utilizar métodos de resolución con tiempos cercanos a *real-time*. Esto limita notablemente las alternativas que se pueden utilizar a tal propósito, únicamente siendo viable entre los métodos de resolución clásicos la utilización de los algoritmos heurísticos constructivos. Ya que dichos algoritmos garantizan una solución, la cual se construye de forma directa.

Sin embargo, particularizar algoritmos heurísticos para la resolución de problemas de combinatoria es una labor tediosa que requiere de una experiencia específica en el ámbito de aplicación. Además la solución ha de ser capaz de adaptarse al entorno dinámico que supone la infraestructura. Por tanto, la capacidad para automatizar este proceso es de vital importancia para conseguir una orquestación inteligente de la red. Para este propósito, parece razonable la utilización de Machine Learning. En concreto, esta tesis hace uso de **Neural Combinatorial Optimization**, una línea de investigación que emplea Reinforcement Learning para aproximar soluciones de forma rápida a problemas de combinatoria discreta.

La propuesta de solución está alineada con la tendencia de incluir Inteligencia Artificial (AI) en la gestión y orquestación de la red 5G (e.g., gestión de recursos radio, gestión de movilidad, gestión de red, gestión de servicios). Se sigue para ello la actividad del Study Group de ITU-T sobre Machine Learning en Redes Futuras incluyendo 5G.

El potencial de la solución técnica depende de varios factores como: la garantía de cumplimiento de los acuerdos de servicio (SLA), la capacidad del método para obtener soluciones en un tiempo delimitado, o la utilización de tiempos de entrenamiento razonables que permitan reevaluar el modelo cuando se detecte un cambio de la infraestructura.

Desarrollo

La tesis aquí presente hace uso de Machine Learning para afrontar problemas de optimización combinatoria, en particular, de Neural Combinatorial Optimization (NCO). NCO permite inferir de forma autónoma un método de resolución para problemas de combinatoria basándose en la experiencia que un agente obtiene interactuando con el problema. La ventaja de esta solución reside en que una vez el modelo embebe las características del problema, este método permite obtener soluciones aproximadas de forma rápida, basándose en la extrapolación de comportamientos que han sido positivos durante el aprendizaje.

En los problemas de combinatoria, las técnicas de aprendizaje supervisado no son aplicables, ya que dichos métodos requieren el cálculo previo de un número elevado de soluciones óptimas, algo inviable en problemas NP-hard. Por ello, NCO hace uso de Reinforcement Learning para este propósito. Reinforcement Learning es un tipo de aprendizaje autónomo que explora el problema y refuerza de forma positiva las combinaciones que resultan en una mayor recompensa. Para lo cual solamente es necesario definir una función de coste.

Para desarrollar dicho modelo, el problema de «VNF Placement» se modeliza como un proceso de decisión de Markov (MDP) ante el cual un agente recoge una cadena de servicio y de forma iterativa computa las probabilidades condicionales de posicionamiento para cada elemento basándose en el aprendizaje previamente adquirido (ver Fig. 0.1). A tal fin, el agente requiere gestionar secuencias para poder abordar servicios de tamaño variable. Los avances en Sequence Learning obtenidos recientemente gracias a campos en expansión como Neural Machine Translation han

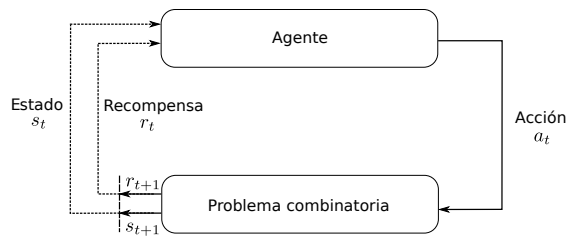


Fig. 0.1.: Esquema para resolución de problemas de combinatoria mediante Reinforcement Learning.

contribuido a la generación de estructuras de redes neuronales que pueden ser aplicadas para este propósito.

El agente desarrollado construye soluciones al problema de forma incremental. Este método supone un avance sobre las redes encoder/decoder utilizadas habitualmente en NCO y provenientes de Sequence-to-sequence Learning. En concreto, se formula el problema siguiendo el framework clásico de un proceso de RL, dónde la solución se compone iterativamente basándose no solamente en la definición del problema sino también en la evolución de la misma en el entorno. Esto permite al agente comprobar la evolución de la solución y tomar cada decisión de placement con una mayor certeza sobre los estados intermedios del problema. Esto supone una ventaja respecto a los anteriores modelos encoder/decoder, ya que estos recurrían únicamente a la memoria interna almacenada en decodificación para tal propósito.

En concreto en la tesis se han evaluado dos modelos para generar el agente: (a) un modelo basado en redes recurrentes (RRN) y (b) un modelo basado completamente en mecanismos de atención sobre la propia secuencia. En el primer caso se hace uso de redes Long Short-term Memory (LSTM) para adquirir una representación de la definición del problema. Mientras que en el segundo caso, se utiliza Transformer network para tratar la secuencia de entrada sin necesidad de recursión. Sino que se codifica la posición relativa de cada elemento de la secuencia y se establece un mecanismo de atención que permite a la red focalizar sobre los elementos que más influyen en cada acción.

En este trabajo se han evaluado técnicas de aprendizaje provenientes de las dos ramas principales de investigación en Reinforcement Learning: Policy Optimization y Temporal-Difference Learning, incluyendo la hibridación entre ambas que generan los algoritmos Actor-Critic. En cuanto a la estrategia seguida en la solución final se trata de un algoritmo Monte-Carlo Policy Gradients with a self-competing baseline. Los mecanismos basados en Policy Optimization se basan en estimaciones *unbiased* pero con alta varianza, algo que dificulta la convergencia del modelo. Para mejorar este comportamiento se suele añadir una red neuronal de apoyo o «critic» que estima la función de valor dependiente del estado. En nuestro caso, el algoritmo empleado consigue disminuir la varianza durante el entrenamiento sin emplear una red neuronal auxiliar. Únicamente la estocasticidad que presenta la política hasta que converge durante el aprendizaje.

El algoritmo de aprendizaje aquí presentado explota la implementación escogida en el modelo. En concreto se ha optado por hacer uso de aceleración por GPU tanto para el computo de la red neuronal como en la evaluación del entorno. Para ello la definición del problema también se ha vectorizado. El algoritmo propuesto se beneficia de la facilidad que aporta esta solución para paralelizar el problema y la utiliza para realizar una aproximación en la distribución de probabilidades de la política, de donde es posible obtener una función de estimación de la recompensa.

Además, hasta la fecha, Neural Combinatorial Optimization no afrontaba problemas con restricciones, o en caso de hacerlo las restricciones se embebían dentro del modelo mediante el uso de máscaras que restringiesen el espacio de acciones con el fin de obtener soluciones factibles. Sin embargo, esto limita de forma explícita los problemas que se pueden afrontar ya que la red utilizada debe de ser diseñada específicamente para el problema, y además esta técnica no puede aplicarse a todo tipo de restricciones. En este respecto se decide emplear formulación proveniente de safe-RL a fin de embeber las restricciones en el algoritmo de aprendizaje, y permitir por tanto al algoritmo tener constancia de la cuantía de insatisfacción de las restricciones para guiar al método hacia una política que las minimice.

Con el fin de mejorar los resultados obtenidos, se evalúan diferentes arquitecturas de redes neuronales y se realiza un estudio de convergencia. La función objetivo en NCO es una función de muy alta dimensionalidad, no-convexa y con ruido proveniente de aproximar el gran espacio combinatorial con pequeños muestreos del mismo. Con el fin de obtener beneficios en su optimización se ha avanzado en las siguientes direcciones:

- Aplicación de técnicas de suavizado de la función de coste. En concreto de técnicas de regularización de la entropía con el fin de mejorar la exploración y poder converger a mejores políticas. En determinadas circunstancias una política con mayor entropía puede suavizar la función y llegar por tanto a conectar óptimos locales. Lo que permite obtener mejores resultados con mayores learning rates.
- Mejorar la estabilidad del aprendizaje evaluando modelos de aprendizaje basados en trust-region optimization. El uso de un paso de tamaño fijo durante el aprendizaje en vanilla Policy Gradients puede llevar a una degradación en la política y por tanto a un mal rendimiento de la red. Trust Region Policy Optimization (TRPO) restringe la divergencia con la nueva distribución que se obtiene en cada iteración en el aprendizaje. También se ha evaluado Proximal Policy Optimization (PPO) basado en un modelo subrogado de la técnica anterior.

El modelo resultante se ha validado en un problema de optimización combinatoria clásico como es el *Job Shop Scheduling Problem*, con el fin de disponer de una referencia contrastada con las que comparar el desempeño del modelo. Las soluciones obtenidas mediante NCO son comparadas con las obtenidas por algoritmos heurísticos, metaheurísticos y los principales solvers de constraint programming en la industria: CPLEX de IBM, Gecode y CP-SAT de Google Or-Tools. La experimentación realizada en el problema prueba la viabilidad de la propuesta a la hora de computar soluciones de manera rápida. En particular, se observa que para el rango bajo-medio de instancias del problema el modelo es capaz de conseguir resultados superiores a los algoritmos heurísticos evaluados. Sin embargo, a medida que el tamaño de instancia crece esta ventaja se desvanece. El modelo requiere de una mayor exploración dado el mayor espacio combinatorial, que de no materializarse (debido al mayor tiempo de aprendizaje requerido) acaba resultando en la inferencia de heurísticas sencillas por parte del agente.

Finalmente, el modelo desarrollado se utiliza para afrontar el problema de «VNF Placement» donde se comprueba la validez de la solución en un entorno de red virtualizado NFV. En dicho problema se verifica la versatilidad del modelo a la hora de afrontar diversas restricciones. Los resultados obtenidos demuestran que ambos modelos propuestos (a) un modelo basado en redes recurrentes (RRN) y (b) un modelo basado completamente en mecanismos de atención, son válidos para afrontar el problema. Los resultados obtenidos en ambos son similares, por lo que se llega a la conclusión de que el codificador secuencial y en general la red neuronal no supone un límite para el rendimiento del modelo, sino que se ha de buscar mejoras en el método de aprendizaje. Con respecto a la comparativa de resultados, el modelo NCO muestra

una capacidad para computar soluciones cercanas a los resultados obtenidos por solvers de constraint programming (aunque en una fracción del tiempo que estos requieren). En particular, para instancias pequeñas el modelo infiere una política cercana a la óptima; y para modelos de mayor complejidad, demuestra ser superior a algoritmos genéticos, los cuales son utilizados como referentes de algoritmos metaheurísticos.

Abstract

The **Fifth Generation of the mobile network (5G)** represents a breakthrough technology for the telecommunications industry. 5G provides a unified infrastructure capable of integrating over the same physical network heterogeneous services with different requirements. This is achieved thanks to the recent advances in network virtualization, specifically in Network Function Virtualization (NFV) and Software Defining Networks (SDN) technologies. This cloud-based architecture not only brings new possibilities to vertical sectors but also entails new challenges that have to be solved accordingly. In this sense, it enables to automate operations within the infrastructure, allowing to perform network optimization at operational time (e.g., spectrum optimization, service optimization, traffic optimization). Nevertheless, designing optimization algorithms for this purpose entails some difficulties. Solving the underlying **Combinatorial Optimization (CO)** problems that these problems present is usually intractable due to their NP-hard nature. In addition, solutions to these problems are required in close to real-time due to the tight time requirements on this dynamic environment. For this reason, handwritten heuristic algorithms have been widely used in the literature for achieving fast approximate solutions on this context.

However, particularizing heuristics to address CO problems can be a daunting task that requires expertise. The ability to automate this resolution processes would be of utmost importance for achieving an intelligent network orchestration. In this sense, Artificial Intelligence (AI) is envisioned as the key technology for autonomously inferring intelligent solutions to these problems. Combining AI with network virtualization can truly transform this industry.

Particularly, this Thesis aims at using **Neural Combinatorial Optimization (NCO)** for inferring end solutions on CO problems. NCO has proven to be able to learn near optimal solutions on classical combinatorial problems (e.g., the Traveler Salesman Problem (TSP), Bin Packing Problem (BPP), Vehicle Routing Problem (VRP)). Specifically, NCO relies on **Reinforcement Learning (RL)** to estimate a Neural Network (NN) model that describes the relation between the space of instances of the problem and the solutions for each of them. In other words, this model for a new instance is able to infer a solution generalizing from the problem space where it has been trained. To this end, during the learning process the model takes instances from the learning space, and uses the reward obtained from evaluating the solution to improve its accuracy.

The work here presented, contributes to the NCO theory in two main directions. First, this work argues that the performance obtained by sequence-to-sequence models used for NCO in the literature is improved presenting combinatorial problems as **Constrained Markov Decision Processes (CMDP)**. Such property can be exploited for building a Markovian model that constructs solutions incrementally based on interactions with the problem. And second, this formulation enables to address general constrained combinatorial problems under this framework. In this context, the model in addition to the reward signal, relies on penalty signals generated from constraint dissatisfaction that direct the model toward a competitive policy even in highly constrained

environments. This strategy allows to extend the number of problems that can be addressed using this technology.

The presented approach is validated in the scope of intelligent network management, specifically in the **Virtual Network Function (VNF) placement problem**. This problem consists of efficiently mapping a set of network service requests on top of the physical network infrastructure. Particularly, we seek to obtain the optimal placement for a network service chain considering the state of the virtual environment, so that a specific resource objective is accomplished, in this case the minimization of the overall power consumption. Conducted experiments prove the capability of the proposal for learning competitive solutions when compared to classical heuristic, metaheuristic, and Constraint Programming (CP) solvers.

Acknowledgement

I would like to thank to Prof. Fidel Liberal and Prof. Bego Blanco for their supervision, without whom this work would not have been possible. I further thank Prof. Josu Ceberio from the Department of Computer Science and Artificial Intelligence of the University of the Basque Country for the guidance and feedback provided throughout this project. And specially, I wish to convey my gratitude to Prof. Martin Takáč from the Industrial and Systems Engineering Department of Lehigh University for the encouragement and technical advice given during my research stay, which really enabled me to progress in this field of knowledge.

Last but not least, I would like to thank to all my colleagues for their support and patience during the preparation of this Thesis. The multiple discussions and meetings formed a great enrichment to my research.

Contents

1	Introduction	1
1.1	Context and motivation of the work	1
1.2	Background on Combinatorial Optimization	3
1.2.1	Exact methods	3
1.2.2	Metaheuristic algorithms	4
1.2.3	Heuristic algorithms	5
1.3	Reinforcement Learning for intelligent heuristics	6
1.4	Neural Combinatorial Optimization for 5G network optimization	6
1.5	Thesis objectives	7
1.6	Thesis structure	8
2	State-of-the-Art of Neural Combinatorial Optimization	10
2.1	Machine Learning for solving CO problems	10
2.1.1	Learning Methods	11
2.1.2	Algorithmic Structures	11
2.2	Neural Combinatorial Optimization	13
2.2.1	Background	13
2.3	Sequence-to-sequence models for NCO	15
2.3.1	Recurrent Neural Networks	16
2.3.2	Recurrent sequence-to-sequence models for solving CO problems	18
2.3.3	Transformer network for solving CO problems	20
2.4	Challenges of Neural Combinatorial Optimization	21
2.5	Conclusions	22
3	Framework for a constrained Markovian NCO model	23
3.1	Baseline Framework	23
3.2	NCO on a Markovian RL approach	24
3.3	Aspects to be considered on NCO	27
3.4	Building blocks of NCO	28
3.4.1	Learning algorithms	28
3.4.2	Proposal for a Markovian neural agent	30
3.4.2.1	Recurrent encoder-based attention mechanism	31
3.4.2.2	Transformer encoder-based attention mechanism	32
3.4.2.3	State-based attention mechanism	33
3.4.3	Constraint management in NCO	34
3.4.3.1	Limitations of Action-masked Networks	34
3.4.3.2	Background on Constrained RL	34
3.4.3.3	Reward constrained policy optimization	35
3.4.4	Optimization algorithms	39

3.4.4.1	Momentum-based optimizers	40
3.5	Conclusions	41
4	Proposal for a model building process and experimentation	43
4.1	Motivation for studying the learning process in a toy CO problem	44
4.2	Methodology for improving the learning process	45
4.2.1	Convergence analysis	45
4.2.2	Enhance exploration strategies in deep RL	46
4.2.2.1	Entropy Regularization	47
4.2.2.2	Intrinsic Rewards as Exploration Bonuses	50
4.2.3	Monotonic improvements with Trust-region optimization	52
4.2.3.1	Trust Region Policy Optimization	53
4.2.3.2	Proximal Policy Optimization	54
4.2.3.3	Implementing Trust-region optimization using OpenAI framework	55
4.2.4	Improving the model with a self-competing strategy	60
4.2.5	Search strategies	62
4.3	Proposed framework for addressing NCO	63
4.4	Experimentation on the Job-shop Scheduling Problem	64
4.4.1	Job-shop Scheduling Problem with limited idle time	64
4.4.2	Particularized models	64
4.4.3	Learning algorithm: PPO with self-competing baseline	66
4.4.4	Results on the Job Shop Problem	66
4.5	Conclusions	70
5	Use-case: Application for 5G real-time placement decision systems	71
5.1	Introduction to Network Function Virtualization	72
5.1.1	Benefits of Network Function Virtualization	73
5.1.2	Description of the ETSI-NFV architecture	73
5.1.3	Network service creation process	74
5.2	VNF Placement Optimization	75
5.2.1	Related work on the VNF Placement problem	76
5.2.2	VNF Placement problem formalization	77
5.3	Experimentation details	80
5.3.1	Model implementation	81
5.3.2	Performance comparison	84
5.3.3	Results	85
5.3.4	Learning and inference times	88
5.4	Conclusions	89
6	Discussion	91
6.1	Performance comparison between recurrent and attentional models for NCO	91
6.2	Graph neural networks applied to NCO	92
6.3	NCO in combination with Tree-Search strategies	92
6.4	RL to enhance Metaheuristic algorithms	93
6.5	Discussion on the ITU-T approach for introducing ML in 5G	93
6.6	Conclusions	94
7	Final conclusions, contributions and broader impacts	95

7.1	Thesis coverage	95
7.2	Main contributions	96
7.3	Final conclusions	97
7.4	Thesis publications	97
7.5	Future Work	99
7.6	Broader Impacts	99
Appendix A Annex: Operations Research		101
A.1	Mathematical Programming (MP) vs Constraint Programming (CP)	101
A.2	Logical Conditions	102
Appendix B Annex: Reinforcement Learning		103
B.1	Markov Decision Process	103
B.1.1	Definition	103
B.1.2	Value Functions	105
B.1.3	Bellman equations for the Value Functions	105
B.2	Planning by Dynamic Programming	108
B.2.1	Policy Evaluation	108
B.2.2	Policy Iteration	108
B.2.3	Value Iteration	109
B.3	Model-Free Learning: Value-based Learning	109
B.3.1	Value-based Prediction	109
B.3.2	Value-based Control	112
B.4	Model-Free Learning: Policy-based Learning	117
B.4.1	Policy Gradients	117
B.4.2	Baseline	119
B.5	Actor-Critic methods	121
Appendix C Annex: Job Shop Problem		124
C.1	Heuristic and metaheuristic algorithms for the Job Shop Problem	124
C.2	Implementation details	124
C.3	Run times	125
Bibliography		131

List of Abbreviations

AC	Actor Critic
AI	Artificial Intelligence
CO	Combinatorial Optimization
CP	Constraint Programming
CPU	Central Processing Unit
DL	Deep Learning
DNN	Deep Neural Network
DP	Dynamic Programming
FFN	Feed-forward Neural Network
GNN	Graph Neural Network
GPU	Graphics Processing Unit
IS	Importance Sampling
JSP	Job-Shop Scheduling Problem
MC	Monte-Carlo
MCTS	Monte-Carlo Tree Search
MDP	Markov Decision Process
ML	Machine Learning
MP	Mathematical Programming
NCO	Neural Combinatorial Optimization
NFV	Network Function Virtualization
NFVI	Network Function Virtualization Infrastructure
NFVO	Network Function Virtualization Orchestrator
NLP	Natural Language Processing
NN	Neural Network
OR	Operations Research
PG	Policy Gradients
PN	Pointer Network
POMDP	Partially Observable Markov Decision Process

QoS	Quality of Service
RCPO	Reward Constrained Policy Optimization
RL	Reinforcement Learning
RNN	Recurrent Neural Network
SBA	Service-Based Architecture
SFC	Service Function Chaining
SGD	Stochastic Gradient Descent
SL	Supervised Learning
SLA	Service Level Agreement
TD	Temporal-Difference
TPU	Tensor Processing Unit
TSP	Travelling Salesman Problem
VM	Virtual Machine
VNF	Virtual Network Function
VNFD	Virtual Network Function Descriptor
VNFM	Virtual Network Function Manager
VRP	Vehicle Routing Problem

“ *Reinforcement Learning is the science of decision making.* ”

— **David Silver**
Head Researcher of Google DeepMind

Contents

1.1	Context and motivation of the work	1
1.2	Background on Combinatorial Optimization	3
1.2.1	Exact methods	3
1.2.2	Metaheuristic algorithms	4
1.2.3	Heuristic algorithms	5
1.3	Reinforcement Learning for intelligent heuristics	6
1.4	Neural Combinatorial Optimization for 5G network optimization	6
1.5	Thesis objectives	7
1.6	Thesis structure	8

1.1 Context and motivation of the work

THE next iteration of the mobile telecommunication network, 5G, is called to introduce a major transformation within its transition into network virtualization. The facilities in network automation this infrastructure provides will enhance network efficiency, in addition to providing network operators the flexibility to quickly react to fluctuations in the traffic demand. In this framework, **real-time decision making systems** are envisioned as a key element for achieving optimal management decisions. Network optimization is pursued at different network stages, e.g. spectrum optimization, service optimization, traffic optimization. In order to achieve this objective, fast approximate solutions are required to be obtained on the underlying **Combinatorial Optimization (CO)** problems that arise under the different network configurations. The ability to automate these optimization processes would be of utmost importance for achieving an efficient network orchestration.

In this regard, one wonders whether the outstanding results that Artificial Intelligence (AI) has recently obtained in strategic thinking can be applied to intelligent network orchestration. Particularly, this work seeks to build a model that autonomously learns from the environment interacting with it, and uses the experience inferred to produce solutions on novel network scenarios. This ability to generalize intelligent decision would be used for optimizing the network utilization. To this end, **Reinforcement Learning (RL)** [108], an specific area of Machine Learning (ML), is considered. RL is a reward oriented technique whose aim is to explore an environment and learn from its own experiences how to lead the agent to maximize the return of its actions.

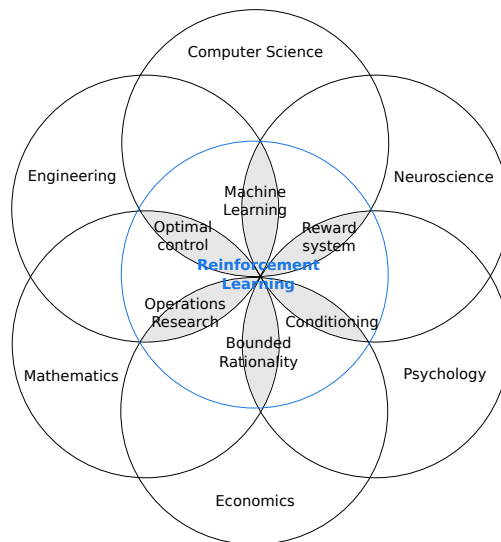


Fig. 1.1.: Applicability of Reinforcement Learning to different fields of study. Recreated from [103].

RL is an increasingly important technology for developing highly-capable decision systems in complex environments. It has proved to offer a viable path to solving hard sequential decision-making problems that cannot currently be solved by any other approach. For example, it has achieved superhuman performance in complex games as Go [105], real-time strategy games as Starcraft [119] and Dota [19]; but also in other complex domains as controlling robotics [100] or serving optimized content to users [43]. Although this technology is still far to serve as an "off-the-shelf" solution, latest RL algorithms are reported to be successfully transferred from one task to another with no task-specific changes and little to no hyperparameters tuning. This is the ultimate goal of artificial intelligence, obtaining an General AI capable of adapting to a wide variety of problems without requiring a specific configuration.

The success achieved in RL has aroused the interest to build models for solving problems of a wide variety of disciplines. In the end, it is a fundamental science that studies the optimal way to make decisions. As RL makes almost no assumptions on the problem setting or its structure, it can be applied in practically all settings, and forces designed algorithms to be adaptive to many kinds of challenges. As depicted in Fig. 1.1, RL is being used to undertake decision-making problems that traditionally the different sciences have approached particularizing resolution models to better suit their goals.

In the particular interest of this Thesis, RL has applicability in Operations Research (OR), specifically for achieving rapid approximations on Combinatorial Optimization (CO) problems. CO is the science concerned with finding an optimal or close to optimal solution among a finite collection of possibilities. CO problems are present in a vast number of real applications: e.g., computing optimal routes, scheduling problems, supply chain or in the scope of this work, network optimization. So far, combinatorial problems have been approached from the applied mathematics and computer science perspective. In that sense, the resolution methods traditionally used to tackle these problems can be divided into exact methods, that guarantee finding optimal solutions; and heuristics/metaheuristics, that trade off optimality for computational cost. **RL adds a new approach to these problems, by providing a method that autonomously learn solutions to the problems without requiring any hand-engineering reasoning.** This makes RL a compelling option with the potential to become an important milestone on the path toward approaching these problems.

1.2 Background on Combinatorial Optimization

Combinatorial Optimization (CO) is the science that studies finding the optimal solution from a finite set of discrete possibilities. An instance of a combinatorial problem is defined by a set of candidate solutions (i.e. the search space Ω) and an objective function. Solving such a problem requires finding the solution that maximizes or minimizes the given objective function. In practice, certain combinatorial problems are considered computationally intractable. In terms of complexity, these problems are usually NP-hard [41]. Under the assumption that $P \neq NP$ ¹ there is no algorithm that in a deterministic manner, obtains optimal solutions consistently for all inputs in a reasonable time.

Historically, combinatorial problems were approached with exact algorithms that guarantee the optimality of the solution. Conversely, for NP-hard problems exact methods do not run in polynomial time, and thus, for large instances these methods are no longer a feasible option. Today, many of the combinatorial problems studied in Operations Research are complex problems classified, according to the theory of complexity, in this category. Therefore, an algorithm to efficiently solve them has not been discovered yet. For this reason, an extensive work has been done in OR to find approximation methods that better suit these problems (see Fig. 1.2). Nowadays, there are three main approaches to such problems:

- exact methods,
- metaheuristic algorithms,
- and heuristic algorithms.

1.2.1 Exact methods

As mentioned, only for some combinatorial problems it is possible to find efficient exact algorithms. For example, the Shortest Path Problem, under some assumptions, can be solved by the Dijkstra or Bellman-Ford algorithms, obtaining optimal solutions in polynomial time. However, this is not the case for problems classified as NP-hard. For these problems, exact methods have an exponential computational cost, so that the time to solve the problem grows exponentially with the number of variables. Several paradigms can be used to obtain exact solutions at least in theory, these are Mathematical Programming and Constraint Programming. However, in practice, these methods are executed until a good enough solution is obtained.

Mathematical Programming (MP)

Mathematical Programming (MP) requires that the problem is classified in a well-defined mathematical category such as Linear Programming (LP), Mixed Integer Linear Programming (MIP), Quadratic Programming (QP), etc. For problems that fall in one of these categories a mathematical program is formulated. This program consists on a set of decision variables, an objective function to maximize or minimize, and a set of constraints equations. This paradigm relies on specific search algorithms that are used depending on the problem classification. Currently, there are commercial solvers (e.g. IBM Cplex, Gurobi, AMPL, OPL, etc.) that facilitate this resolution process.

¹The complexity class P contains all decision problems that can be solved exactly in polynomial time on a deterministic machine. The NP class makes reference to "Nondeterministic Polynomial" problems that can be solved exactly in polynomial time on a nondeterministic machine. It is generally assumed that $P \neq NP$. An optimization problem X is NP-hard if there is a NP-complete decision problem Y that can be reduced to X in polynomial time (Garey et al., 1979) [41].

A representative algorithm used in MP is the Simplex algorithm that, although exponential in the worst case, is routinely used to solve LPs in polynomial time. Other algorithms based on implicit enumeration as Branch-and-Bound, can be very effective for practical MIP. This method begins by finding the optimal solution through the relaxation of the problem without the integer constraints (via standard linear or nonlinear optimization methods). If in this solution, the decision variables with integer constraints have integer values, then no further work is required. If one or more integer variables have non-integral solutions, the Branch-and-Bound method chooses one such variable and branches, creating two new subproblems where the value of that variable is more tightly constrained. These subproblems are solved and the process is repeated, until a solution that satisfies all of the integer constraints is found.

Constraint Programming (CP)

Using Constraint Programming (CP) the problem is also expressed in a declarative fashion as in mathematical programming. However, CP technology is based primarily on computer science fundamentals, such as logic programming and graph theory. In contrast to MP, which is based on numerical linear algebra.

CP is used when there are complex logical and arithmetic relationships between decision variables. E.g., in sequencing or scheduling problems. CP works first reducing the set of possible values of the decision variables that satisfy all the constraints by using logical, graph-theoretic, arithmetic, and other arguments. When the deduction of some values from the decision variable's domain are not possible, this information is propagated through the constraints perhaps enabling further deductions. Various search strategies are also used until a value is assigned to every decision variable, that is, until a solution is found. After a first solution is found, the search proceeds to find further solutions with better objective values. CP does not provide a lower bound on the solution. Finally, as in the previous case, there are commercial solvers that abstract the implementation of this method (e.g., Gecode, IBM Cplex-CP, CP-SAT, etc).

Conclusions on exact methods

MP and CP methods deal with the underlying non-convexity of the optimization problem these programs present relying on exhaustive search. Both methods use a divide and conquer approach, where the problem to be solved is recursively split into sub-problems by fixing values of one variable at a time. The main difference between them lies in how each node of the resulting problem tree is obtained. MIP usually solves a linear relaxation of the problem and uses the result to guide search (e.g., in Branch-and-Bound search). Whereas in CP, logical inferences based on the combinatorial nature of each global constraint are performed.

In practice, it might happen that exact algorithms provide good approximations or even optimal solutions in polynomial time to intractable problems of large size. However, the opposite can also happen, **exact algorithms may require prohibitive time for addressing small size instances.**

1.2.2 Metaheuristic algorithms

Conversely, metaheuristic algorithms are multi-purpose methods that provide a general definition on the components of the resolution process and their interactions. Their fundamental is based on proposing randomly generated solutions. These methods transform existing solutions into new candidates through methods such as permutation, mutation, crossover, etc. Hoping that the resulting solution continues satisfying the constraints, but with a better objective value. These are iterative processes that are repeated until a sufficiently good solution is found. Among the

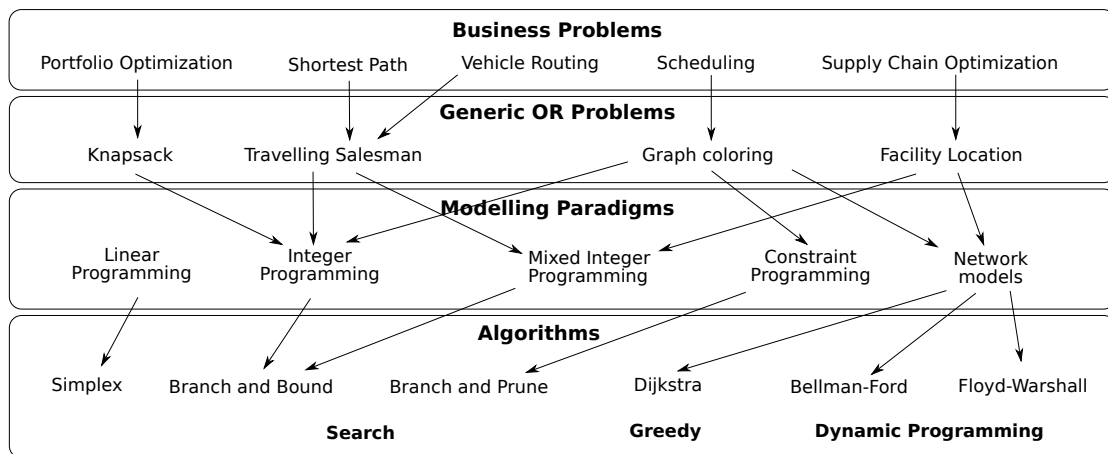


Fig. 1.2.: Four layers of abstractions on Operations Research.

well known metaheuristics we cite: Genetic Algorithms, Local Search, Tabu Search, Simulated Annealing, Ant Colony Optimization, Swarm Optimization, etc.

Nevertheless, metaheuristics present a mayor drawback. Despite their good results in solving optimization problems, **when the number of restrictions is high and the feasible region of solutions is small, they tend, in some cases, to be ineffective.**

1.2.3 Heuristic algorithms

A practical approach to many intractable problems of large size are heuristic algorithms. These algorithms are able to produce acceptable solutions using reasonable resources. In practical terms, they offer a good alternative when it is not required to find the optimum solution, but only good enough approximation. Heuristics are based on handcrafted rules that exploit some specific feature of the problem. Therefore, the human experience to code the rules that fit the problem is key for the results. In this case, the quality of the obtained solutions, that is, the effectiveness of the algorithm, highly depends on the quality of the rules themselves, hence on the ability of the engineer that designs them. Heuristics can be used either to directly achieve problem-specific solutions or to guide search strategies that are specifically designed for the problem.

A possible classification on heuristic algorithms is the following:

- **Constructive heuristics:** operate iteratively selecting the "best" subset of a given set of elements. These methods start from an empty set and iteratively add one element to the solution applying some specific selection criterion. For example, if the selection criterion is some "local optimality" (e.g., the element providing the best improvement the objective function), one obtains the so called *greedy heuristics*. The basic feature of such construction approaches is the fact that, in principle, the selection made at a certain step influences only the following steps, that is, no backtracking is applied.
- **Approximation algorithms:** are heuristic methods able to provide a performance guarantee. With approximation heuristics it is possible to formally prove that, for any instance of the CO problem, the obtained solution will never be worst than the optimal solution (which may be unknown) over a specified threshold. E.g., approximation algorithms that provide a guarantee on the Bin Packing are: Next-fit, First-fit, Best-fit, etc.

1.3 Reinforcement Learning for intelligent heuristics

While heuristic algorithms have reached impressive success for some otherwise impossible problems, they present a major drawback, they lack of a theoretical background. These algorithms rely on handcrafted rules for making decisions that otherwise would be too expensive to computationally obtain. Thus, Machine Learning (ML) looks like a natural candidate to automate such decisions in a more principled way.

In particular, a novel line of research has succeeded tackling combinatorial optimization problems using Neural Networks (NN) and Reinforcement Learning (RL). This discipline received the name of **Neural Combinatorial Optimization (NCO)** [15]. Particularly, NCO automatically discovers greedy heuristics that produce direct solutions on combinatorial problem without human intervention [83, 65, 35]. This approach has the potential to be applicable across many combinatorial optimization tasks only requiring minor hand-engineering to adapt the neural models. Recent works have proved that this technique is able to learn competitive heuristics when compared to classical alternatives. Unlike classical heuristics that are typically expressed in the form of rules that define how to obtain the solutions, NCO interpret these rules as policies used to make decisions on the problem. NCO parametrizes this policy using a neural network that is trained to learn intelligent behaviours by means of a RL approach. One of its main benefits, is that once the model embeds information of the problem, solutions are computed in real-time. Solutions cannot be proven to be optimal, but they improve as more information from the whole combinatorial space is evaluated and used to reinforce the agent.

Due to these reasons, NCO is of special concern to build the real-time decision-making systems envisioned in this Thesis. Many decision-making systems seek to obtain intelligent decisions in conditions of uncertain or constantly changing environments, and are required to do so in a limited period of time. To this end, these time-critical systems usually rely on handcrafted heuristics for this purpose. Even though these heuristics work well, if the problem statement changes slightly, they need to be revised. For this reason, the recent idea of autonomously learning competitive heuristics that adapt to the problem without human intervention is gaining attraction in Operations Research.

1.4 Neural Combinatorial Optimization for 5G network optimization

In the particular focus of this Thesis, Neural Combinatorial Optimization (NCO) is used for optimizing the telecommunication network. Due to the growing demand of mobile services, the current mobile infrastructure is rapidly evolving towards more efficient virtualized architectures. In this scope, the automation of the network orchestration is one of the key aspects pursued. Whereas existing fourth-generation (4G) networks are based on a reactive conception, the network is reconfigured after adverse situations occur and normally it is done throughout handwritten rules. The 5G network, is called to introduce a major transformation within its transition into intelligent network management. In this direction, Artificial Intelligence (AI) is envisioned to allow to the 5G network a proactive and predictive management, enabling a more efficient network optimization.

In these schema, NCO has a great potential for dealing with problems in which existing solutions require of hand-engineering, for complex problems for which there is no good solution using

traditional approaches, for adaptation to fluctuating environments and in general to notice the patterns that a human cannot perceive. Due to this reasons, current network management is envisioned to evolve from complex heuristics that can be hard to maintain, to intelligent systems capable of automatically learn from previous interactions to predict intelligent decisions.

Specifically, this Thesis seeks to design a real-time decision system to optimize the placement of the Virtual Network Functions (VNF) that compose the virtual services in this architecture. This work will result in a more efficient network utilization while ensuring the Service Level Agreements (SLA). This maximizes the mutual benefits between network operators and service providers.

At the time of this publication, the **ITU-T Focus Group on Machine Learning for Future Networks including 5G (ITU-FG-ML5G)** is actively working to standardize use-cases, data formats and ML-aware architectures embeddable in the telecommunications networks. The majority of ML optimization strategies that this study group and in general the scientific community research on this area fall into supervised learning. Henceforth, the models studied are focused in regression and classification, approaches that present important implementation drawbacks in this context. On the other hand, NCO presents a novel approach in which these decision models can be automated without human intervention. No optimal labels are required to train the models, instead the model discovers the environment and reacts accordingly. This is of special importance in these highly dynamic environments in which human intervention is impractical or has a high cost.

1.5 Thesis objectives

The main purpose of the Thesis can be divided into the following objectives and sub-objectives:

- **Define a NCO model for solving combinatorial problems.** Particularly, this work focuses on combinatorial problems that can be represented as sequences. Many combinatorial problems are classified into this category and have a sequence representation (e.g., the Traveler Salesman Problem (TSP), Bin Packing Problem (BPP), Vehicle Routing Problem (VRP)).
 - **Achieve a comprehensive understanding on the different RL strategies.** Different approaches can be used in RL to perform a model-free learning. In this sense, a study in the different alternatives is required to validate them in the scope of CO.
 - **Identify the building blocks for the NCO model.** Analyze the different elements that form the NCO model for further discuss on the best suitable alternative. In this sense, several aspects need to be considered when addressing CO problems (e.g., high dimensionality, deterministic behaviour, observability). The aim is to identify how these premises affect on the different elements of the model.
 - **Consider constraint management on the NCO framework.** So far, existing works in NCO have built specific agents that ensure feasibility, or have relied on masking schemes to prevent undesirable solutions. This work aims at presenting an strategy that allows to apply NCO to general constrained combinatorial problem. Expanding the problems that can be addressed using this technology.
- **Define a methodology for addressing CO problems.** In order to consider a suitable strategy, it is crucial to estimate the learning capabilities of the models, detect the weak

points and explore enhance methods to better optimize the model. To this purpose, different learning strategies need to be studied and benchmarked. This methodology provides a reference for the learning capabilities of the models and would serve as a baseline from where to draw conclusions applicable on further problems.

- **Select the most suitable learning strategy and validate the solution.** Propose a candidate architecture and evaluate it in the experimentation. To this end, a classical CO problem is used. This is because these are well-documented problems in the literature that enable to benchmark the solution against proven heuristics, metaheuristics and commercial solvers.
- **Application of the NCO model to the use-case "real-time VNF placement decision systems in 5G environments".** NCO is aimed at building a real-time decision-making system in the scope of intelligent systems to orchestrate the 5G network. Particularly, the VNF placement problem seeks to optimize the placement of service chains within the virtualized infrastructure envisioned in future communication networks. One of the main challenges in this technology is the optimal resource placement within the infrastructure. The **placement of the virtualized network functions** in the cloud environment and the network embedding can be formulated as a mathematical optimization problem concerned with a set of constraints that express the restrictions of the network infrastructure and the Service Level Agreements (SLAs). The development of **real-time decision-making systems** able to serve rapid approximations to that combinatorial problem would be of utmost importance to optimize the network utilization, and thereby, contribute to the implantation of intelligent network orchestration.

1.6 Thesis structure

The manuscript is organized as follows:

Chapter 2 — State-of-the-Art of NCO

In this Chapter, the state of the art ML strategies for solving CO problems are presented. The different learning methods and structures in which ML can be used for addressing CO problems are reviewed. In the following, NCO is introduced together with the background on this technology. This is done emphasizing the sequence-to-sequence models that traditionally have been used in the literature to cope with these problems. Finally, the drawbacks of this setting together with the challenges that this technology faces are shown.

Chapter 3 — Framework for a constrained Markovian NCO model

This Chapter introduces the formalism used in RL. Here, the Markovian model we argue in this Thesis is presented. The benefits of the proposed model are remarked when compared to previous sequence-to-sequence models. Also, an strategy for dealing with constraints is introduced. For this purpose, reward constraint policy optimization technique is applied to CO problems. This is done including an analysis on the resulting objective function. Finally, the different learning algorithms that enhance the proposal as well as an analysis on the different optimization strategies is included.

Chapter 4 — Proposal for a model building process and experimentation

This Chapter analyzes the learning process in the proposed Markovian model. To this end, we motivate the use of a toy combinatorial problem in order to extract conclusions on the different learning strategies. In this sense, a convergence analysis is done, regularization techniques are applied and also, a self-competing strategy is proposed to improve the convergence on the method.

The resulting model is evaluated in the classical the Job-shop Scheduling Problem (JSP), and constrained variants of it. A complete experimentation is performed against different heuristic, metaheuristic and Constraint Programming (CP) solvers.

Chapter 5 — Use case: Application for 5G real-time placement decision systems

This Chapter presents a use-case for the designed model. This is done in the scope of network optimization, specifically for optimizing the placement of virtual network functions inside a virtualized 5G infrastructure. For this purpose, the standardized ETSI-NFV architecture is described to further formulate the optimization problem mathematically. The resolution of this problem is done following the strategy previously analyzed in Chapter 4. From the results here obtained we conclude that the analyzed solution is a viable alternative for building real-time decision systems under the mentioned assumptions.

Chapter 6 — Discussion

In this Chapter different alternative research lines for addressing CO problems are discussed. An analysis of the different works and future strategies is presented indicating whether the specific proposal would be a valid option for the use case addressed in this Thesis. Finally, the work of the ITU-T study group focused in ML applied to 5G is here exposed. An analysis of the different case of studies this entity is carrying on is done and a motivation for implementing the RL alternative we argue in this work is suggested.

Chapter 7 — Final conclusions, contributions and broader impacts

This final Chapter summarizes the most important aspects covered in this Thesis. Here, the contributions and the final conclusions are remarked. This Chapter ends emphasizing the broader impacts of the NCO technology and the repercussion of this work on the OR community.

State-of-the-Art of Neural Combinatorial Optimization

Contents

2.1	Machine Learning for solving CO problems	10
2.1.1	Learning Methods	11
2.1.2	Algorithmic Structures	11
2.2	Neural Combinatorial Optimization	13
2.2.1	Background	13
2.3	Sequence-to-sequence models for NCO	15
2.3.1	Recurrent Neural Networks	16
2.3.2	Recurrent sequence-to-sequence models for solving CO problems . . .	18
2.3.3	Transformer network for solving CO problems	20
2.4	Challenges of Neural Combinatorial Optimization	21
2.5	Conclusions	22

OPERATIONS Research (OR) started during the second world war as an initiative to combine mathematics and computer science to obtain an advantage on military decisions. Nowadays, it is widely used in the industry, being specially relevant in sectors such as transportation, supply chain, energy or finance. As previously analyzed in Section 1.2, several alternatives have been traditionally used to address CO problems depending on the limitations that the problem imposes (e.g., computational time, optimality gap, etc). However, not all of them are suitable for building the real-time decision-making system we seek in this work. Several of the mentioned techniques are used during the Thesis to perform a benchmark on the results, although not all meet the tight resolution times that an interactive system requires. **This fact restricts the alternatives to address the problem, only remaining as viable option the use of greedy heuristics.** Greedy heuristic algorithms operate making the locally optimal choice at each stage, and although they do not provide any guarantee on the solution, represent the best suitable alternative for the decision-making systems concerned in this Thesis.

However, designing heuristic algorithms for combinatorial optimization can be a daunting task that requires expertise in the problem. Because of this, the recent idea of automating this process is an appealing objective in the OR community. Recent attempts at leveraging ML to solve combinatorial problems have shown the viability of this alternative. Due to this reason, this is the line of research explored on this work.

2.1 Machine Learning for solving CO problems

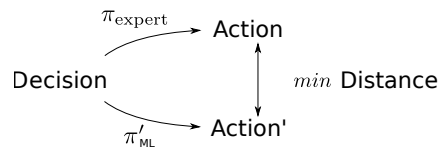
This Thesis is focused on researching algorithms that autonomously learn how to address combinatorial problems. To this end, Machine Learning (ML) looks like the natural candidate to cope with such objective. The challenge that ML pursues is to find an algorithm that performs well on problem instances used in the learning process but also on unseen instances. This is known

as **generalizing**. This ability that ML models have to extrapolate from examples seen during the training is one of the strengths of this discipline and it can be used to create models that contribute at different stages on the CO resolution process.

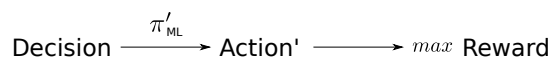
2.1.1 Learning Methods

Usually, two different motivations justify the use of ML to address CO problems. In the first case, the researcher assumes theoretical or empirical knowledge on the problem and wants to automate the decision process by approximating the solutions with ML. The second use is motivated by the fact that a positive behaviour on the problem is unknown and a solution on the problem needs to be learnt from scratch. Thus, a trial and error reinforcement process is used to explore the problem and find good behaviours. In greater detail:

- **Supervised Learning** (SL) strives for learning the decisions that an expert does on the problem. The function that has to be learnt for making the decisions is called the policy π . A function that given all available information about the problem, returns the action to be taken. In SL, the policy function π is learnt using demonstrations on the results. Examples can be found on the field of MIP [12], assisting on Branch-and-Bound searches [6, 61, 42] or directly to generalize from previously computed solutions [120].



- **Reinforcement Learning** (RL) on the other hand, is used to autonomously learn on the problem. In that case, the model seeks to explore the space of decisions, and learn from experience the best performing behaviour. This is done through a reward mechanism, a learning method based on reinforcing the actions that accumulate the higher outcome. E.g., in [28] RL is used to support a local search procedure. Search-methods often are guided by heuristics, here alternatively a policy is learnt to perform the local rewrite of the current solution based on inherited intuitions.



It is critical to understand that in SL, the policy is learnt through targets provided by an expert; whereas in RL, the policy is learnt from a reward signal. In SL, the agent is taught what to do, whereas in RL, the agent is encouraged to explore the problem and improve its policy to maximize the returns on the process.

2.1.2 Algorithmic Structures

ML can be incorporated as part of the solution algorithm at several stages on the resolution process. For example, in the more direct case the ML-model can be trained to obtain complete solutions, but also it can be used to support traditional OR algorithms. In this section, the different approaches in which an ML-model can be used throughout OR algorithms are presented:

- **End-to-end learning:** represents the most direct approach to address CO problems. End-to-end learning seeks to train a model that directly outputs the solution from the problem definition (see Fig. 2.1). Directly addressing CO problems presents a technical challenge, as an approximation for the whole combinatorial space is required. Due to this reason, it has not been until the rise of Deep Learning (DL) that this approach has become a feasible option [120].

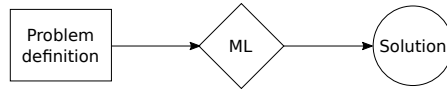


Fig. 2.1.: The ML model is used to directly obtain complete solutions on the problem.

- **Learning to configure OR algorithms:** in many cases, using only a ML model to tackle the problem may not be the most suitable approach. Instead, ML can be used to infer additional information to take decisions on CO algorithms (e.g., to perform operational decisions or select the configuration of the hyperparameters). A good selection on these parameters can dramatically change the performance of the optimization algorithm. Hence, the OR community is engaged to automate these decisions using ML (see Fig. 2.2). The idea behind this method is that a good configuration can be inferred for different cluster of similar problem instances. Examples of this method can be found in [66, 22].

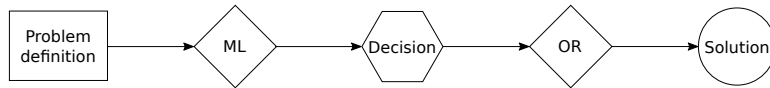


Fig. 2.2.: The ML model is used to configure an OR algorithm.

- **Learning alongside OR algorithms:** ML can be also incorporated as part of an OR algorithm. In that case, the ML model is repeatedly called throughout the execution, assisting in lower level decisions on the problem resolution (see Fig. 2.3). The ML model is used by the CO algorithm to make intermediate decisions taken during the iterations of the algorithm. E.g., this is the case of the Branch-and-Bound algorithm for MILPs, where the task of selecting the branching variable can be performed by a ML model [127].

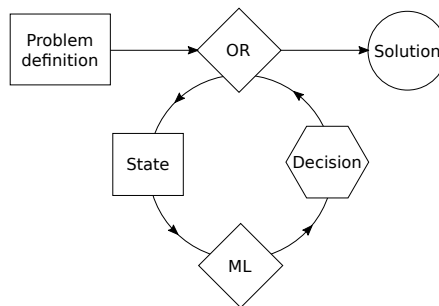


Fig. 2.3.: ML alongside the optimization algorithms. The ML model can be incorporated into the OR algorithm to take decisions through the problem resolution.

According to the classification exposed, the model pursued on this Thesis can be categorized as follows. This Thesis is focused on learning greedy heuristics without relying on human intervention, to this end, this work seeks to learn a model that obtains direct solutions on the problem relying only on autonomous explorations. Thus, Reinforcement Learning (RL) is the

technology that is required to explore and learn on the problem. Also, to meet the tight time limits that interactive systems require, an end-to-end learning must be pursued to obtain direct solutions on the problem. As mentioned, the use of Deep Learning (DL) has brought a great success on this area. DL is a ML sub-field focused on building large parametric non-linear approximators. DL excels when applied in high dimensional spaces, for this reason, it offers the perfect match to build the models necessary for solving CO problems. Currently, there is line of research that joins DL and RL to obtain direct solutions on CO problems. This technique was introduced in (Bello et al., 2016) [15] and the authors named this field of study *Neural Combinatorial Optimization*.

2.2 Neural Combinatorial Optimization

Neural Combinatorial Optimization (NCO) [15] is a recent line of research that combines Deep Learning (DL) and Reinforcement Learning (RL) to directly infer solutions on combinatorial problems. In this framework, the model autonomously learns an heuristic that achieves rapid approximate solutions on CO problems. To this end, the inferred heuristic generalizes from instances of the problem space where it has been trained. The effectiveness that NCO has shown in recent works has motivated us to follow this approach to build the decision-making system pursued in this Thesis.

Specifically, NCO estimates a neural network model to describe the relation between the space of instances of the problem \mathcal{X} and the solutions for each of them \mathcal{Y} . The parameters of the model are estimated iteratively by means of an RL approach. Particularly, during the learning process the model takes instances from the problem space, and uses the reward (cost value in our problem) obtained from evaluating the solution returned by the neural network to improve its accuracy. A important requirement to apply RL is that evaluating the objective function must not be time-consuming, a feature that combinatorial problems inherently present.

2.2.1 Background

The use of neural networks for solving combinatorial optimization problems dates back to (Hopfield et al., 1985) [54]. The authors applied the Hopfield-network for solving instances of the Traveller Salesman Problem (TSP). Nevertheless, the application of neural networks on combinatorial problems was limited to small scale problem instances due to the available computational resources at that time. It has been in the last few years with the rise of DL that this topic has again attracted the attention of the OR community.

Recently, (Vinyals et al., 2015) [120] trained a Deep Neural Network (DNN) to solve the euclidean TSP using supervised learning. They proved that a neural network is able to parametrize a competitive policy also in domains with large action spaces as it is the case of most real-world combinatorial problems. To this end, they introduced the Pointer Network (PN), a neural architecture that enables permutations of the input sequence, a requirement for dealing with the TSP. Despite their positive results, using supervised learning to solve combinatorial problems is not trivial, as acquiring a training set implies having a large amount of solved instances.

In (Bello et al., 2016) [15] the framework of NCO was presented, and RL was implemented for the first time to solve combinatorial problems. The authors took the PN introduced by Vinyals and utilized an actor-critic architecture to solve the TSP problem. That work proved that it is possible to learn competitive heuristics without human intervention. Although, they had to apply

Tab. 2.1.: Relation between the neural network architectures and the structure of the information they are optimized to process.

Type of NN	Information Structure
Fully Connected NN	Arbitrary
Convolutional NN	Spatial
Recurrent NN	Sequential
Graph NN	Relational

heavy sampling and searching techniques at inference to improve the solution generated by the neural network itself.

Using RL, optimal labels are not required to train the model, instead the rewards that the model obtains from evaluating the solutions are used to optimize the policy. In the original approach [15], Policy Gradients [108] method was used to perform a stochastic gradient descent to estimate the parameters of the neural network. Once the agent converges on the learning process, given an instance of the problem to the model, it immediately returns a solution. This solution cannot be proven to be optimal, yet is "close" to it in the sense that it follows the policy that, on average, has obtained the best results.

In (Deudon et al., 2018) [35] the TSP problem was also optimized using the NCO approach. In this occasion, the Transformer Network [116] was used, a top performance architecture in Natural Language Processing. In that work, the greedy output of the neural network is hybridized with a local search to improve the results at inference. Independently, (Kool et al., 2018) [65] also presented the same architecture, yet with improvements in the decoder as well as in the training mechanism. This strategy allows them to be competitive without applying search strategies at inference.

All the works presented so far have major similarities, they are based on sequence-to-sequence models, architectures originally designed for supervised learning. In these models, the solution is decoded at once based only on a single interaction with the problem. Conversely, (Nazari et al., 2018) [83] approached the Vehicle Routing Problem (VRP) as a Markov Decision Process (MDP). Specifically, they construct the solution in sequence of decisions made interacting with the environment. Such approach allows to focus on how the environment evolves during the construction of the solution. Strategy that enables also to deal with stochastic versions of the problem as the model is able to react to changes in the definition while the solution being computed. This method is also aligned with [125], work done in the context of Constraint Satisfaction Problems (CSP), where a value function is learnt using Q-Learning to obtain direct solutions.

Recently, combinatorial problems with graph representation as the Minimum Vertex Cover or the Maximum Cut have been also addressed using RL (Khalil et al., 2018) [60]. In this case, Graph Neural Networks (GNN) are used to embed the structure of the problem (Table 2.1 summarizes the different structures of information neural networks are designed to process) and performs a Q-learning strategy to incrementally construct the solution. In this occasion, no post-processing at inference is used. Another example of a problem redesigned to use GNNs is [72]. The authors address the continuous stochastic job arrival using a fixed length waiting queue to feed a DNN that schedules the pending individual tasks. In [73], this work is evolved, being the jobs to directed graphs and uses a GNN under the same assumptions. Showing the benefits of GNN embedding in terms of problem flexibility and results.

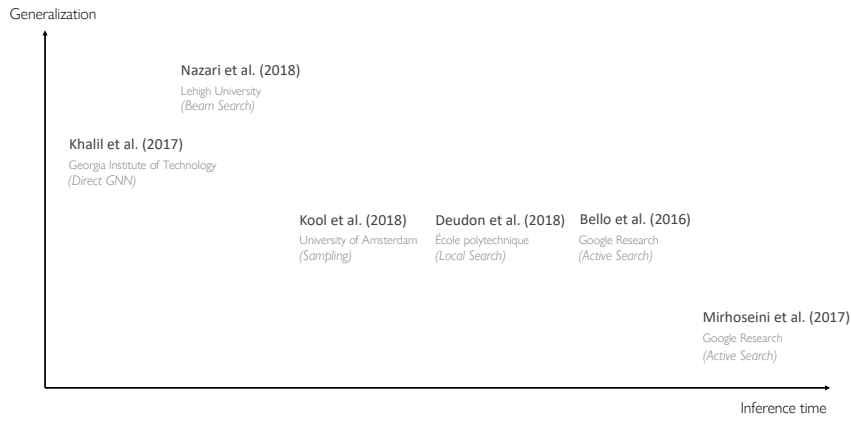


Fig. 2.4.: Comparison in terms of generalization and inference time of the most relevant works in NCO. For the purpose of this Thesis, the better generalization the model presents in the shortest inference time the better.

A different approach that worth a mention is (Mirhoseini et al., 2017) [80]. In that work, a resource allocation problem is solved using RL, however the objective it pursues is quite different than the intended in this Thesis. It seeks to explore the best solution for a single instance of the problem. This is done overfitting the model to the instance, which implies that the model does not generalize and has to be retrained for a every single instance. This differs form goal in this Thesis, that is to infer an heuristic able to generalize solutions from a problem distribution.

Finally, the main works in NCO arranged according to their ability to generalize and the inference time they require are depicted in Fig. 2.4. The more complex problems the model is able to generalize in the less inference time, the better. As mentioned, some of these works require some post-processing at inference, what increases the time to obtain the final solutions. This is also a variable that has to be taken into consideration on the model selection.

2.3 Sequence-to-sequence models for NCO

Traditionally, sequence-to-sequence models have been the core design used in NCO to address combinatorial problems. In problems that fit into this category, an instance is defined by a sequence x that represents the input to the model; similarly, the output corresponds to another sequence y that defines the solution. E.g., in the TSP problem, a sequence of cities have to be visited minimizing the distance traveled. An instance of the TSP is therefore defined by the sequence of cities, and the solution by another sequence indicating the order in which the cities have to be visited. In this problem, the solution corresponds to a permutation of the input elements, therefore both sequences have the same length. However, sequence-to-sequence models are a more powerful tool, they are also able to address sequences of different lengths. E.g., in the VRP problem, a vehicle has to find the optimal route in order to deliver to a set of customers. In this case, the vehicle may visit the depot many times, so if the deliveries are not done at once, the length of the solution would be larger that number of cities.

Sequence-to-sequence models are used in NCO to directly learn a match from an input sequence representing an instance of the problem to an output sequence which corresponds to the solution (see Fig. 2.5). When sequence-to-sequence models are addressed using RL, the whole solution is evaluated at once on the problem to compute the reward. In ML, these edge

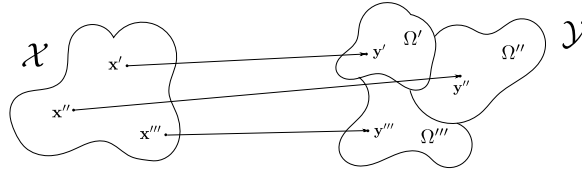


Fig. 2.5.: Space representation for combinatorial problems. The NCO model generalizes from the instances \mathbf{x} from the problem space \mathcal{X} , in order to match them to solutions \mathbf{y} in the output space \mathcal{Y} . Notice that the search space for each problem instance Ω , is a subspace of the whole output space the model covers. This is because the masking scheme limits the search space for each particular instance of the problem.

cases in which the process only requires of a single interaction to with the problem are called "bandit problems".

The complexity in sequence-to-sequence models resides in designing a DL model that produces solutions on the desired problem space Ω . E.g., Pointer Networks were designed as a particular sequence-to-sequence model to produce permutations, and thus, they are able to cope with combinatorial problems as the TSP or the Knapsack problem, but no others. Therefore, one of the major concerns in this research line is to design problem specific models that fit in the problem definition. In sequence learning, these models can be classified in the following groups:

- **Recurrent models:** models that seek to transform sequences based on recurrent encoder-decoder architectures. They use Recurrent Neural Networks (RNNs) as memory cells to preserve in their hidden internal state a representation of the input from where compute the output sequences. E.g., the Pointer Network (Vinyals et al., 2017) [120] belong to this group.
- **Transformer models:** transform one sequence into another using an encoder-decoder architecture that relies solely on attention mechanisms. Since these models do not contain recurrence to preserve the order of the sequence, they develop a positional encoding mechanisms. E.g., the Transformer network (Vaswani et al., 2017) [116], which is currently the state-of-the-art model for natural language processing (NLP).

2.3.1 Recurrent Neural Networks

Recurrent Neural Networks (RNNs), in addition to capture the non-linearities and deal with high dimensions as feedforward networks (FFNs), they preserve the internal state throughout a feedback loop that connect their past decisions. Adding memory to neural networks has the purpose of retrieving the information that sequentially feeds the model.

RNNs are formally dynamical systems and therefore, can be described as a set of equations that describe the state transition and the output equation:

$$\begin{aligned} c_t &= g(c_{t-1}, x_t) && \text{State transition} \\ h_t &= f(c_t) && \text{Output equation} \end{aligned}$$

where $g : \mathbb{R}^K \times \mathbb{R}^I \rightarrow \mathbb{R}^K$ is a state transition function and $f : \mathbb{R}^K \rightarrow \mathbb{R}^J$ the output function, $x_t \in \mathbb{R}^I$ represents the external inputs, $c_t \in \mathbb{R}^K$ the inner states and $h_t \in \mathbb{R}^J$ the output of the system, with $I, J, K \in \mathbb{N}$. The state transition is a mapping from the internal hidden state of the

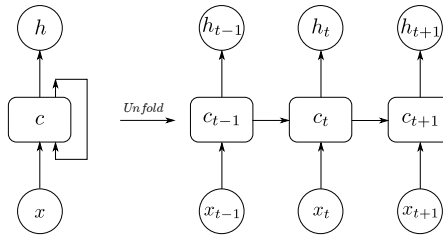


Fig. 2.6.: Recurrent Neural Network (RNN) represented as an unfolded dynamical system, with input x , internal state c and output h .

system c_{t-1} and the external inputs x_t to the next state c_t . The output equation computes the observable output h_t out of the current state c_t .

Recurrent networks have two sources of input, the present and the recent past, which are combined to determine the output. That sequential information is preserved in the recurrent network's hidden states, which allow embedding inter-temporal dependencies. Theoretically, in the recurrent framework an event in state c_t is explained by a superposition of external inputs $\{x_t, x_{t-1}, \dots\}$ from all the previous time steps.

The technique to train RNNs is called backpropagation through time or BPTT, and is a generalization of back-propagation from feed-forward networks (FFN). Conceptually, BPTT works by unrolling all input timesteps. Each timestep has one input, one copy of the network, and one output. Errors are then calculated and accumulated for each timestep. The network is rolled back up and the weights are updated. Spatially, each timestep of the unrolled recurrent neural network may be seen as an additional layer given the order dependence of the problem, and the internal state from the previous timestep is taken as an input on the subsequent timestep (see Fig. 2.6).

Vanishing and exploding gradients

Unfortunately, RNNs suffer from a well-know problem called vanishing gradients [52], which prevents them from being accurate on large dependencies. This is a difficulty that appears in training Deep Neural Networks with gradient-based learning methods. In such methods, the parameters are updated proportionally to the partial derivative of the cost function in each iteration of the training. The problem is that through many-layered networks, the gradient will be vanishingly small, effectively preventing the weight from changing its value. The further we move backwards on the model, the bigger or smaller our error signal becomes. This means in the case of RNN that the network experiences difficulty in memorizing elements from far away in the sequence and makes predictions based on only the most recent ones.

To solve the vanishing gradients on recurrent neural networks, a particular cell was designed, the long short-term memory (LSTM) [53]. LSTM cells help to preserve the error that can be backpropagated through time and layers. By maintaining a more constant error, they allow recurrent nets to continue to learn over many time steps, thereby opening a channel to link causes and effects remotely.

A newest and most effective ways to resolve the vanishing gradient problem has recently been achieved using residual neural networks, or ResNets [49]. ResNets deal with this problem by utilizing skip connections, or shortcuts to jump over some layers. As hypothesized by the author, adding skip connection not only allows data to flow between layers easily, it also facilitates the learning of identity function, thus it is allowed flow without transformations. The network is able

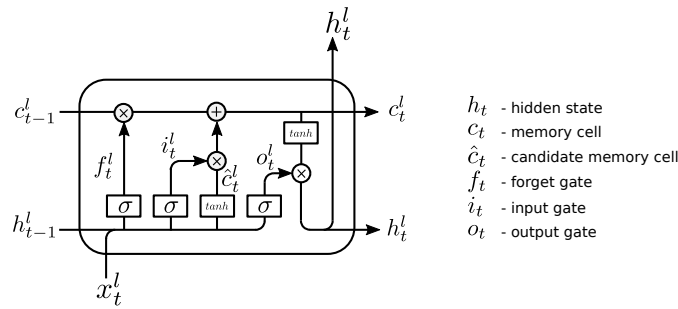


Fig. 2.7.: Internal gate structure of a Long Short-term Memory cell (LSTM). RNNs cells can be stacked to get deeper models, in the figure the l -layer is depicted.

to learn better when having identity mapping by it's side to use, as learning perturbations with reference to identity mappings are easier than to learn function as a new one.

Exploding gradients is another common problem that appear when training RNNs. The gradient of some parameters become saturated on the high end. However, exploding gradients can be solved relatively easily, clipping the gradients at a predefined threshold [85].

Learning Long-Term Dependencies

As mentioned, LSTMs are one of the most common recurrent cells explicitly designed to avoid the long-term dependency problem. These cells have the ability to remove or add information to the cell state, carefully regulating structures called gates. This way they are able to selective forgetting or remembering information from sequence. Otherwise, information that is critical to the task at hand, could be overwritten by redundant or irrelevant information. Gates are composed out of a sigmoid neural net layer and a pointwise multiplication operation. This describes how much of each component should be let through.

Particularly, an LSTM cell (see Fig. 2.7) has three gates: the input gate (i), the forget gate (f) and the output gate (o). The input gate controls the extent to which a new value flows into the cell, the forget gate controls the extent to which a value remains in the cell and the output gate controls the extent to which the value in the cell is used to compute the output activation of the LSTM unit. The internal state of LSTM stored in the memory cell vector (c) and the output of the cell corresponds to the hidden state (h).

2.3.2 Recurrent sequence-to-sequence models for solving CO problems

The sequence-to-sequence reference model studied in this Thesis is based on a recurrent encoder-decoder architecture (depicted in Fig. 2.8). This architecture was introduced in Neural Machine Translation (Sutskever et al. 2014) [107], achieving top scores and relegating previous model based on statistical translation. This architecture is not exclusive for translation, and as it is discussed below, it can be used for general sequence transformations. It can be divided in the following parts:

- An **encoder network**: consisting of a RNN that processes the input sequence and encodes its information into a vector that serves as the “context” to the decoder.
- A **decoder network**: formed by a different RNN trained to predict the output sequence given the feature vector previously computed on the encoder.

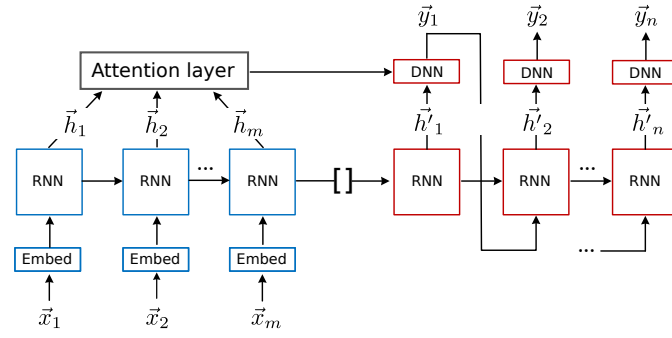


Fig. 2.8.: Recurrent sequence-to-sequence reference model. It includes a sequential encoder (blue) that codifies the sequence that represent the input to the model. And a decoder (red) that uses codification of the instance of the problem to recursively form the solution relying also on its internal memories.

Recurrent sequence-to-sequence models have been extensively used in NCO to learn heuristics on the problem upon the basis of generalization [15, 35, 65, 80]. This means that once the model learns how to interact with the problem, for any instance, it directly computes a solution generalizing from the instances seeing during the learning process. The model receives an instance codified as a sequence $\mathbf{x} = \{\vec{x}_1, \vec{x}_2, \dots, \vec{x}_m\}$ of a variable size m and outputs another sequence $\mathbf{y} = \{\vec{y}_1, \vec{y}_2, \dots, \vec{y}_n\}$ of length n indicating the final solution. The main reason for using sequence models, is because this architecture is able to manage sequences of different sizes without modifying its internals.

As mentioned, learning heuristics on sequence-to-sequence models is a particular case of a "bandits problem" [59], as these models directly seek a solution from the whole combinatorial space. Following this strategy, the model not only presents a large input space, as it has to generalize from all possible instances of the problem \mathcal{X} ; but also a large output space formed by all possible solutions for each case \mathcal{Y} . To effectively learn models on a high dimensional action space using RL, a particular method shall be used, Policy Optimization (see Annex B.4).

Policy Optimization seeks to directly learn the parameters θ of the stochastic policy $\pi_\theta(\mathbf{y}|\mathbf{x})$ that, for a given a instance $\mathbf{x} \in \mathcal{X}$, assign high probabilities to solutions $\mathbf{y} \in \mathcal{Y}$ with lower costs, and low probabilities to those with higher costs. This method is an effective technique used in high dimensional or continuous action spaces [68]. On this premise, Policy optimization technique shall be used in opposition to value iteration methods, which are not viable in this type of problems which large solution spaces.

Specifically, in sequence-to-sequence models the policy distribution π is obtained using the chain rule to factorize the output probability from the decisions taken on the decoding process. The likelihood of an output y_i in the sequence depends not only on the instance \mathbf{x} but also on the previously part of the output sequence $y_{(<i)}$,

$$\pi_\theta(\mathbf{y}|\mathbf{x}) = \prod_{i=1}^n \pi_\theta(y_i | y_{(<i)}, \mathbf{x}) \quad (2.1)$$

Finally, the solutions the model produce are evaluated on the problem to obtain the rewards necessary to reinforce positive solutions and thus, improve the policy. To this end, the cost function shall to be computed. Notice that for the model, the mathematical description of the problem is unknown, it remains as a black-box. It is during the interaction with the problem that

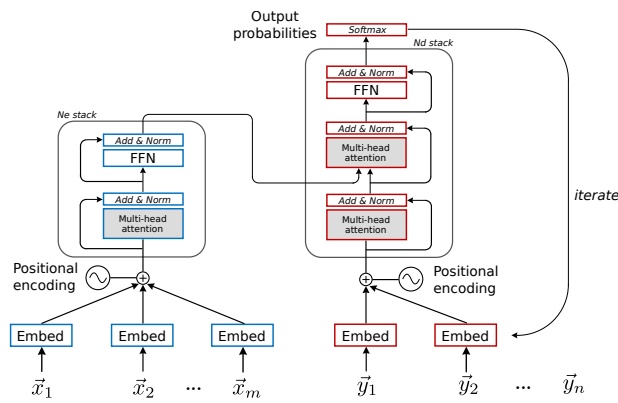


Fig. 2.9.: The Transformer network uses a self-attention mechanism on both the encoder (blue) and the decoder (red) to directly focus on elements of the input and output sequence in the decision making process. This architecture operates iteratively on the decoder, producing one-by-one the elements of the output sequence. In this process the model has direct access to the input throughout multi-headed attention layers.

takes place on the learning process that the model improves its policy to solves the underneath optimization program without even knowing its definition. Subsequently, the problem does not rely on directly optimizing the optimization program, but on optimizing the policy function that generalizes a good strategy.

Attention Mechanisms

One side-effect of sequence-to-sequence learning is that performance comes down when complex relations between sequences need to be learnt. In this sense, attention mechanisms are a useful technique usually applied on those models that enable the decoder to attend to parts of the input sequence on each decision. Focusing just on the part of the sequence that has the relevant information at each step allows to improve the performance. Examples of attention mechanisms are (Bahdanau et al., 2014) [11], or (Luong et al., 2015) [70].

2.3.3 Transformer network for solving CO problems

Recently, neural networks based solely on attention mechanisms have established state-of-the-art performance in sequence modeling. The **Transformer network** (Vaswani et al., 2017) [116] has set a precedent improving existing results in the BLEU¹ score. The Transformer network operates also in an encoder-decoder basis but does not codifies information in an auto-regressive manner. The overall architecture relies on stacked self-attention layers for both encoder and decoder (see Fig. 2.9).

The model codifies the position of each element on the sequence using a positional encoder. This is done using sinusoids of different wavelengths to identify the relative position between the elements of both the input and output sequences. The model uses multi-headed attention layers to attend over the subsequent positions, which in particular is a scaled dot-product attention mechanism. These sub-layers are connected using residual connections [124], which improves the performance allowing to the model to have direct connection between non-contiguous layers. The output produced is passed from the encoder to the decoder, mapping traditional recurrent sequence-to-sequence models. Although, this time every step on the decoder process has the ability to attend directly on every position of the input sequence.

¹The BLEU score is an algorithm for evaluating the quality of text which has been machine-translated between languages.

As preciously mentioned, this sequence-to-sequence architecture has been used in NCO [35, 65]. Although, as will be seen further on, relying on direct connections benefits when compared to recurrence, this model still present a "bandit setting" that does not present the best approach on CO problems.

2.4 Challenges of Neural Combinatorial Optimization

Currently, NCO presents some limitations that narrow the applicability of this technology [17]. The ability to overcome these challenges would be of utmost importance to introduce this method as an useful alternative in the OR portfolio. In the following, the challenges this technology presents are discussed.

Lack of initial conditions

Traditionally, sequence-to-sequence models do not consider the initial conditions in the problem. The input to the model is limited to the instance definition. In case that some initial condition needs to be incorporated in this "*bandits setting*" approach, some tricks have to be used for embedding this information together with the definition of the instance [72]. This present some limitation, for example, usually the dimensionality of the problem definition does not match the initial conditions.

The work presented in the following Chapter 3 contributes in this direction presenting a NCO approach that enables to input the state of the problem (including the initial conditions on the environment) separately from the definition of the problem instance.

Feasibility management

As already noted, RL models do not give any guarantee in terms of optimality, but neither they do with respect to feasibility. So far, in the literature NCO has been used to solve combinatorial problems without dealing with constraints. Existing works overcome this issue building specific models that ensure feasible solutions, or they rely on masking schemes to avoid exploring actions that are a priori known to be unfeasible. In both cases, it is the neural model who guarantees constraint satisfaction. To this end, the neural architectures are designed carefully in order not to break these restrictions. It is the case for instance of the pointer networks (Vinyals et al., 2015) [120] and the Sinkhorn layer (Emami and Ranka, 2018) [39], which are complex architectures designed specifically to compute permutations. Other options as (Nazari et al., 2018) [83] design an specific masking mechanism for the problem that avoids selecting unfeasible solutions. In this regard, in the following Chapter 3 the NCO formulation is broaden to include general constrained combinatorial problems.

Modeling complexity

In ML and in particular in DL, good priors on problems can be used to choose the model that adapts better to the problem. For instance, it is know that convolutional neural networks (CNN) will learn and generalize more easily than others alternatives on image data. The problems studied in CO are different from the ones traditionally addressed in ML, where most successful applications target natural signals. And therefore, the architectures used to learn good policies in CO might be very different from what is currently used for other purposes. Currently, deep learning architectures are evolving to specifically tackling these problems. As pointed, techniques such as parameter sharing made it possible for neural networks to process sequences with RNNs

or, more recently, to process graph structured data through GNNs (Khalil et al., 2017) [60]. Structures that are of uttermost importance to represent CO problems.

Scalability

Finally, the scalability is also a challenge that these models currently present. If a model is trained on instances up to a certain size and the model wants to be tested in larger problem instances, a challenge exists in terms of generalization. In current models a degradation in the performance is observed in case the size of the problem is increased beyond the examples seen during training (e.g., Vinyals et al., 2015 [120]; Bello et al., 2017 [15]; Khalil et al., 2017 [61]; Kool and Welling, 2018 [65]). To tackle this issue, one may retrain the model on larger instances. However, except for very simple ML models and strong assumptions about the data distribution, it is not possible to know the computational complexity required to retraining the problem.

2.5 Conclusions

Traditionally, in the literature sequence-to-sequence models have been used in NCO to address combinatorial problems. To this end, the model is configured in a "bandit setting", where the agent computes the solution at once based on the instance definition. In this sense, the neural architectures used for this purpose mostly arise from Natural Language Processing (NLP). Particularly, in this Chapter recurrent sequence-to-sequence and fully attentional models are discussed. However, these models were designed first for being applied in SL, and when used under a RL approach some drawbacks emerge e.g., specific learning methods have to be used, in this case, for problems with huge or continuous actions spaces.

In addition, the sequence-to-sequence approach discussed in this Chapter presents some technical drawbacks that limits its implementation on general CO problems. For example, this architecture does not consider how initial conditions are introduced into the model. This is due to the fact that the state of the environment is not explicitly considered. Alternatively, one may conclude that CO problems can be tackled under a classical RL framework. Under this approach, the solution is iteratively constructed in a series of actions the agent performs taking into account how the solution evolves on the problem. In this sense, the model benefits from incorporating intermediate steps on the resolution process. Following this premise, this approach to the problem becomes more natural and the algorithms used to address CO problems are not restricted regarding the huge output combinatorial space Ω .

Framework for a constrained Markovian NCO model

Contents

3.1	Baseline Framework	23
3.2	NCO on a Markovian RL approach	24
3.3	Aspects to be considered on NCO	27
3.4	Building blocks of NCO	28
3.4.1	Learning algorithms	28
3.4.2	Proposal for a Markovian neural agent	30
3.4.3	Constraint management in NCO	34
3.4.4	Optimization algorithms	39
3.5	Conclusions	41

As previously discussed in Chapter 2, Neural Combinatorial Optimization (NCO) enables to autonomously learn greedy heuristics on combinatorial problems. For that purpose, NCO has mainly relied on sequence-to-sequence models, presenting the problem in a "bandits setting" in which the goal is to select the best answer from the whole search space Ω .

This Thesis proposes a different strategy in order to enhance this technology to obtain better results. Unlike the original NCO proposal, here, the Markov Decision Process (MDP) formulation in the problem is considered. The solution is therefore iteratively constructed in a series of decisions taken based on the intermediate states obtained during the resolution process. This gives to the model a better understanding on how the solution evolves on the problem, and therefore, improves the quality of the results obtained.

Also, NCO presents some limitations that prevents it from being directly used in constrained problems (as previously mentioned in the Section 2.4 — Challenges), which severely limits this technology from being used in real-world problems. So far, NCO has been used in the literature to solve combinatorial problems without dealing with the feasibility of the solutions [80]. This is possible in case the neural model guarantees constraint satisfaction. Nevertheless, this cannot be achieved in many combinatorial problems. In this work, the feasibility problem is addressed presenting a strategy for applying NCO to virtually any constrained combinatorial problem.

3.1 Baseline Framework

This Chapter presents the baseline framework for addressing NCO problems under a MDP formulation. This framework evolves from the previous "bandits setting", where policy-based methods were required to learn on the huge combinatorial space Ω , to a MDP formulation based on episodic RL interactions (see description in Fig. 3.1). In this context, solutions are iteratively constructed taking partial decisions on the problem. Unlike in the previous approach, here decisions are chosen from an action space that it is explorable. This expands the resolution methods applicable on these problems to value-based and model-based learning strategies.

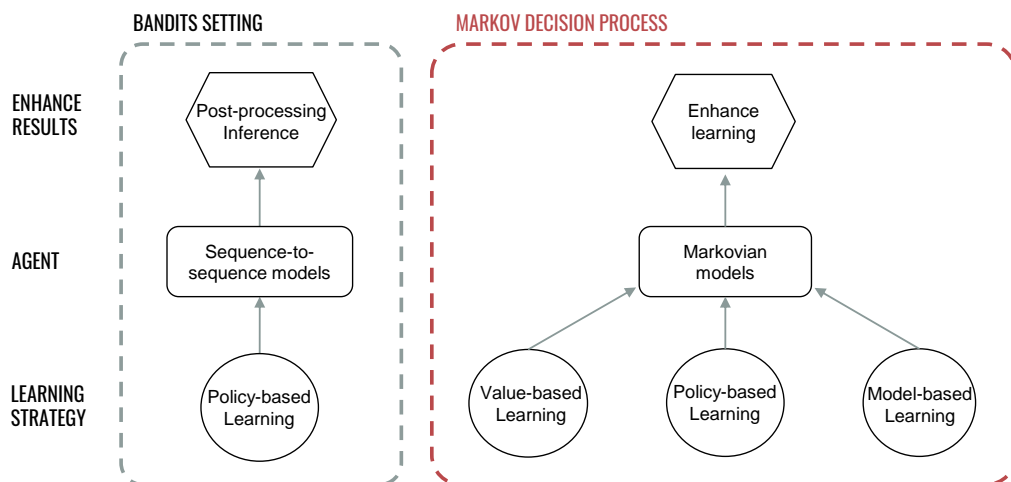


Fig. 3.1.: Baseline framework. Evolution from a sequence-to-sequence approach used to address CO problems in a "bandit-setting" to an iterative RL approach based on the Markov Decision Process (MDP) formulation.

In addition, unlike previous works that aim to reduce the optimality gap performing computations over the obtained solutions at inference time (e.g., an active search [15], local search [35], a sampling strategy [65] or a beam search [83]), our work focuses on enhancing the learning process in order to infer better solutions directly from the neural model. Post-processing strategies usually add a significant computation time at inference. Hence, they are not a candidate strategy for the purpose of this Thesis. Conversely, we aim to improve the policy as much as possible in order to infer better results.

In order to present this approach, in the following section the components that intervene in the RL schema are defined to further introduce how NCO could benefit from this approach.

3.2 NCO on a Markovian RL approach

RL provides a general approach to learn the optimal behaviour in unknown environments. In RL, the problem setting makes almost no assumptions about the model or its structure and usually supposes that the environment is given in the form of a black-box. RL tackles these problems introducing them on the mathematical basis of MDP (see Annex B.1). In this Thesis, we argue that combinatorial problems can be defined as decision processes presenting the Markov property (Definition 3.2.1), and so, are suitable to be addressed using general RL techniques.

Definition 3.2.1 *Markov property:*

«A model presents the Markov property if the transition function that defines the process depends solely on previous state and the last chosen action, being independent of all previous history.»

To argument this proposition, let us introduce first the elements that intervene in a RL model:

- The **environment** corresponds to any kind of dynamical system, e.g. stock market, revenue management, or as in this case, combinatorial problems. The environment evolves based on its history of states and the actions performed by the agent. For each interaction at every time-step t , the system evolves from the current state S_t to the next state S_{t+1} returning a reward R_{t+1} that serves as an evaluation criteria for the action selection.

- The **agent** represents the controller of the system. The agent uses the state information S_t to interact with the environment by performing an action A_t . In return it retrieves a reward signal R_{t+1} , which it is used to improve its policy.
- The **Actions** influence the development of the environment. Actions can be bounded or limited by the problem state, and according to the problem setting the actions are the decisions taken in the resolution process.
- The **policy** π defines the actions taken by the agent. Instead of pursuing a one-step optimization, the policy determines the optimal behaviour with regard to a given overall objective. According to (Sutton and Barto, 1998) [108] "the policy is the core of a Reinforcement Learning agent in the sense that it alone is sufficient to determine its behaviour".
- The **reward** function specifies the overall objective of the Reinforcement Learning problem. It depicts the immediate reward R_{t+1} the agent receives for performing a certain action A_t at a given system state S_t . Consequently, it defines the desirability of an event for the agent. Generally a simple immediate reward is only of minor interest because high immediate rewards might lead to low ones in the future. Instead, one is usually interested in the reward associated to the whole episode.

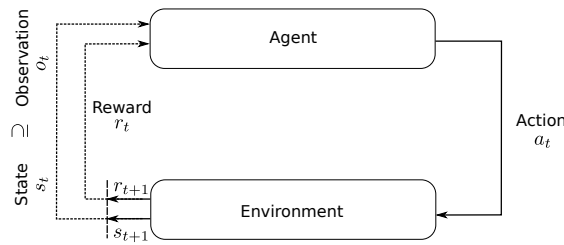


Fig. 3.2.: Reinforcement Learning schema: An agent iteratively interacts with an environment by carrying out an action A_t based on its observed state information O_t , which can be partially observable or fully observable from the state representation S_t . In return it retrieves a feedback in form of a reward R_{t+1} , which it uses to improve its policy and thereby increase its future sum of rewards.

These elements are all put together in the RL feedback loop depicted in Fig. 3.2. Formally, let therefore be $\mathcal{S} \subseteq \mathbb{R}^I$ the environmental state space with states $S_t \in \mathcal{S}$, and $\mathcal{A} \subseteq \mathbb{R}^J$ the control or action space with control or action parameters $A_t \in \mathcal{A}(S_t)$ (with $I, J \in \mathbb{N}$ and $t = 0, 1 \dots T$). In a general decision process, the current state of the problem is defined by the sequence of historical states and actions experienced in the episode, the trajectory: $\tau_t = A_0, S_0, \dots, A_t, S_t$.

$$S_{t+1} = f(\tau_t) \quad (3.1)$$

If the problem presents the *Markov property*, then the state at present time-step S_t contains all required information for the agent to determine its next action A_t . Once the state is known, the history may be thrown away. The state is a sufficient statistic of the future. Decisions made on the current state S_t can be optimal. Its decision rule and respectively policy is therefore a direct mapping from the observed state S_t to the next action A_t . The system can then be described by the following equation,

$$S_{t+1} = g(S_t, A_t) \quad (3.2)$$

with $g : \mathbb{R}^I \times \mathbb{R}^J \rightarrow \mathbb{R}^I$ being an arbitrary, non-linear function.

Throughout this Thesis, CO problems are presented in a MDPs approach in which the sequence of actions that the agent denotes conform the solution to the problem. In other words, the resolution of these problems is presented as a constructive process in which at

each step the agent computes a part of the solution. In this context, **the Markov property is satisfied taking as the state representation the evolution of the partial solution on the problem environment**. E.g., in the Bin Packing problem the representation of the state during the resolution process can be defined by the state of the bins at a time-step t .

This description of the learning process for CO problems introduces several key assumptions. Firstly, combinatorial problems are managed in a temporal time-space. However, they do not have any temporal interpretation, in this case the «time» only represents the order of the elements that form the solution. Secondly, it is assumed that the problem environment incorporates some reward function as the indicator of success. This reflects the Reward hypothesis (Hypothesis 1), also referred to as Reinforcement Learning hypothesis.

Hypothesis 1 *Reward hypothesis:*

«In RL all goals and purposes can be described as a maximization of the expected value of the cumulative sum of the received rewards.»

The objective is therefore to solve CO problems by learning a policy π , that for an instance of the problem, produces the series of actions (conform the solution) that maximizes the objective function. It basically considers the combinatorial problem as a black-boxed environment the agent interacts with, receiving as the outcome for each interaction a reward signal. When dealing with combinatorial problems, it seems natural to think that only at the end of the episode, a reward equivalent to the evaluation of the complete solution on the cost function would be received. However, some techniques [60] enable to define intermediate rewards computing the change in the cost function from evaluating partial solutions. This process will be further detailed, but it useful to clarify that rewards can be received during the whole interaction. Therefore, during the learning process the agent gets positive and negative rewards which it uses to optimize its action selection i.e., the control policy π .

The corresponding reward function is denoted by $R_t := R(S_t) : \mathcal{S} \rightarrow \mathbb{R}$. Generally the particular immediate reward R_t is not of major interest as a corresponding action might lead to low rewards in the long run, at the end of the episode. Therefore, the return function is defined as the accumulated reward over time with a discount factor γ :

$$G_t \doteq \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad \forall t > 0$$

The discount factor $\gamma \in (0, 1]$ is used in RL processes with far-away horizon or no horizon to delimit the present value of future rewards: a reward received k time steps in the future is worth only γ^{k-1} times what it would be worth if it were received immediately. A γ close to 0 leads to a "myopic" evaluation, whereas a γ close to 1 leads to "far-sighted" evaluation. This factor is also used to avoid infinity returns in cyclic Markov processes. In the case of CO problems the discount factor is not required. These problems are deterministic and the number of steps in an episode for achieving a solution are finite. This leads to the conclusion that these problems do not need to fix an horizon in the process, thereof in this Thesis $\gamma = 1$, the return G_t corresponds to the sum of the rewards.

Finally, the policy π is discussed. Unlike in sequence-to-sequence models where the policy is a direct match between an instance of the problem and a solution; here, in a MDP the policy represents the probability of selecting an action given a state on resolution process. Therefore, it establishes a relation between states and actions $\pi : \mathcal{S} \rightarrow \mathcal{A}$ (see Annex B.1.1).

Definition 3.2.2 In a Markov Decision Process a stochastic policy π is defined as a probability distribution over the actions given the states,

$$\pi(a|s) \doteq \mathbb{P}(A_t = a | S_t = s)$$

In conclusion, during the learning process the agent discovers the dynamics of the CO problem interacting with it. The model infers a policy, a plan ahead strategic thinking that constructs the solution iteratively making a series of actions on the problem. The goal is therefore to learn the action selection that result in the best outcome for the problem. In this approach, the action space at every step in the resolution process $A_t \rightarrow \mathbb{R}^J$ becomes explorable, this will have possible implications in the resolution method.

Learning of the optimal control policy

The objective is therefore to find the optimal control policy (details in optimal control are available in Annex B.3). Learn the policy that maximizes the return (in the case of CO problems achieves the optimal solution). This is equivalent to determining the optimal action selection $\pi^* : \mathcal{S} \rightarrow \mathcal{A}$, which achieves the best action A_t based on the approximated inner state of the system S_t ,

$$A_t = a \sim \pi^*(\cdot | S_t = s) \quad \forall t > 0$$

3.3 Aspects to be considered on NCO

The challenges to address in RL can be classified according many different factors depending on the setting utilized and the underlying objective. In the following, the aspects to be considered when solving CO problems are pointed out. Indicating where the existing methods present drawbacks or even fail to produce satisfactory results. CO problems here addressed show three important characteristics:

- high-dimensionality,
- fully observable,
- and deterministic nature.

High-dimensionality

Deep Learning has enabled RL to scale to decision-making problems that were previously intractable. It has allowed to deal with problem with high-dimensional state and action spaces. As mentioned in Section 2.3.2, combinatorial problems present a high-dimensional observations, as the model is structured to perform well on any problem sampled from the distribution of instances. But also, in case of being formulated in a "bandits setting" they present a high-dimensional action-space, the corresponding to the whole search space Ω . To address CO problems with this setting, sequence-to-sequence models have been traditionally used, and Policy-based methods were required on the learning process. This is due to policy-based methods are the ones suitable for large action spaces (see Annex B.4).

In the case that CO problems are described in a MDP approach, the options to address them widen. Notice that, under this assumption, although the state space is similarly huge, the action space becomes explorable. In this proposition, the action space corresponds to the actions available during each time-step (e.g., in a Bin Packing at each time-step the actions to place an item

correspond to the number of bins). This extends the available learning algorithms that can be used to address the problem.

Observability

CO problems are by nature defined as fully observable, $s_t = o_t$. Under this premise, the agent can determine the state of the system at all times. For example, in a the Bin Packing, the state of the system, that is, the position of all items on the bins, is available so that the agent can make an optimal decision. Fully observable problems can be modeled as a MDP and can be addressed with **memory-less agents**, this is opposed to partial observable models or POMDP. In which the agent requires to rely on memories to complement the observed state o_t , as it does not fully represent the state of the problem s_t .

Deterministic

CO problems are generally deterministic. This means that the environment that models the problem behaves the same for similar inputs. However, it comes also natural to this markovian NCO approach to address stochastic combinatorial problems. E.g., [83] shows how to handle a stochastic version of the VRP. In this work random customers with random demands appear over time, during the resolution process. In case a customer changes its demand value or relocates to a different position, the model can automatically adapt the solution. Using classical heuristics for VRP, the entire distance matrix must be recalculated and the system must be re-optimized. In contrast, using NCO framework the model adapts to the new situation, and only one feed-forward pass of the network is required to update the routes based on the new data.

3.4 Building blocks of NCO

So far, the formalism used in RL has been introduced, the MDP model, and briefly the challenges that this technology faces have been noted. In the following, we will point out the different components required to build the learning model. In this work, different strategies are evaluated on the NCO schema to further select the appropriate approach that would be used in the final solution. The selected strategy would be the base from where to design the models used in the experimentation. In the following sub-sections, the different components required to build the learning model are discussed. These components are:

- the learning algorithm,
- the neural agent,
- the strategy for dealing with constraints
- and the optimization algorithm.

3.4.1 Learning algorithms

Classically, there exists two main approaches to address RL problems: methods based on value functions and methods based on policy optimization. There is also a hybrid actor-critic approach, which combining both, value functions and policy search, has achieved state-of-the-art results in domains as motion control or strategic thinking [81]. In the following these approaches are explained:

- **Value-based methods** (Annex B.3.2): estimate the value of being in a given state and use this information to determine the control policy. The state estimation is done by making use of the Bellman's Equations. Most popular Value-based RL algorithms include SARSA-

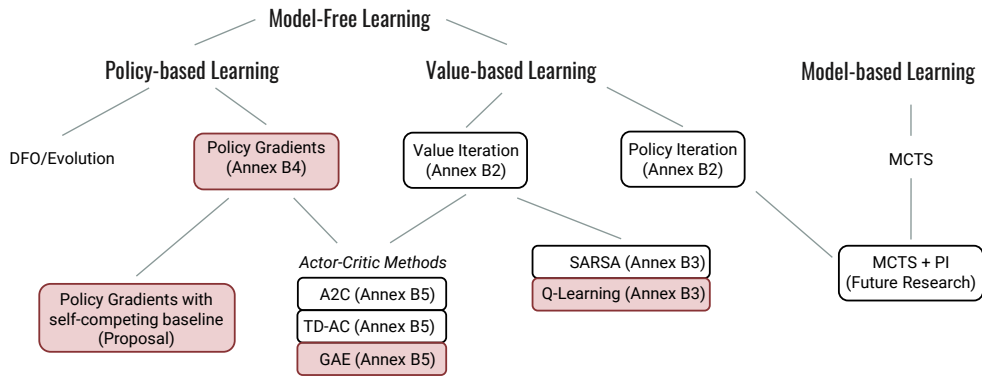


Fig. 3.3.: Reinforcement Learning strategies. The different learning algorithms are classified into the major learning approaches: policy-based learning, value-based-learning and model-based learning. For each method we indicate whether the algorithm is analyzed on this work (highlighted in red color).

learning (Annex B.3.2) and Q-learning (Annex B.3.2), which differ in their TD-targets approximations, that is, the target value to which Q-values are recursively updated by a step size at each time step.

- **Policy-based methods** (Annex B.4): in contrast to the Value-based methods, Policy-based methods directly update the policy without looking at the value estimations. They are slightly better than value-based methods in the terms of convergence, solving problems with continuous or high dimensional spaces, and in addition are capable of inferring stochastic policies. There are several alternatives to optimize these models, however as argued in Annex B.4, gradient-based methods is the of choice that present better result for most deep RL algorithms [38].
- **Actor-critic methods** (Annex B.5): these methods try to reduce the high variance that policy-based methods suffer. The high variance problem is particularly exasperated in problems with long horizons or high-dimensional action spaces. To this end, Actor-critic methods combine value function estimators with an explicit representation of the policy (see Fig. 3.4). The actor learns a policy by using feedback from the critic (value function) instead of relying in noisy estimates. In doing so, these methods trade off the variance of policy gradients with the bias that value-based methods introduce [81].

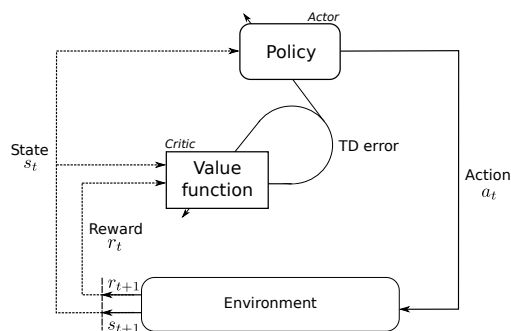


Fig. 3.4.: In actor-critic methods, the actor receives the state from the environment and chooses the action to perform. At the same time, the critic (value function) receives the state and reward resulting from the previous interaction. The critic uses the Temporal-Difference (TD) error calculated from this information to update itself and the actor estimator.

In this Thesis, learning strategies from the described methods will be evaluated on combinatorial problems. Particularly, the algorithms studied are highlighted in red in Fig. 3.3. The result on this analysis will be described in the following Chapter 4. In addition, we discuss an alternative model-based strategy that even though it has not been evaluated in this work, it could be a promising line of research. This alternative combines Monte-carlo Tree Search (MCTS) with Policy Iteration (Annex B.2) to perform a model-based learning that reduces the breadth of the search strategy. Further discussion on this strategy is available in Chapter 6.

3.4.2 Proposal for a Markovian neural agent

The second element required for building the model is the neural agent. Particularly, this section presents the fundamentals of the Markovian agent argued in this Thesis. But first, the nomenclature used in the model is introduced. Let us represent each instance of the problem as a static sequence of feature vectors $\mathbf{x} \in \mathcal{X}$, where \mathcal{X} stands for the whole distribution of instances the model needs to learn an heuristic. The sequence \mathbf{x} defines the problem instance and does not change during the iterative resolution process. Let \vec{d}_t be the vector that represents the state of the environment. \vec{d}_t represents the dynamic part of the input and evolves iteratively as partial decisions are made. The concatenation of those feature vectors at every time-step t represents the input to our model, the state vector $\vec{s}_t \doteq \{\mathbf{x}, \vec{d}_t\}, t = 0, 1, \dots, T$.

This work argues that sequence-to-sequence models compute the solution for combinatorial problems without interacting with the environment. Those models receive an instance and build a solution directly based only on the hidden state the decoder stores (Fig. 3.5a). By contrast, the proposed agents (Fig. 3.5b - 3.5c) present similarities with traditional RL models used for solving fully observable Markov processes. In those cases, the encoding layers constitute the major part of the design, since computing the action distribution is directly done by DNNs that do not require to store or embed any previous information.

Remark 1: This Thesis argues that combinatorial problems can be defined as a fully observable MDPs. Since the solution is iteratively built interacting with the problem, partial solutions can be evaluated to give to the agent a reference on how the solution evolves on the problem. In that perspective, the recurrent decoder used on sequence-to-sequence models can be substituted by a memory-less DNNs. This benefits the results as accessing the fully observable state of the problem is more reliable than doing on memories.

As indicated in Chapter 2, recurrent sequence-to-sequence architectures (Sutskever et al., 2014) [107] compute a mapping from an input sequence $\mathbf{x} = \{\vec{x}_1, \vec{x}_2, \dots, \vec{x}_m\}$ of a variable size m to an output sequence $\mathbf{y} = \{\vec{y}_1, \vec{y}_2, \dots, \vec{y}_n\}$ of different length n . These models use a RNN to read the input sequence, one element at a time, to obtain large fixed-dimensional vector representation. Then, another RNN is used to extract the output sequence from that feature vector (Fig. 3.5a). This model is usually enriched with a differentiable attention mechanism that allows neural networks to focus on different parts of the input.

The neural model proposed in this work changes the recurrent decoder in favour of a DNN that iteratively computes the solutions based on the intermediate state of the environment (Fig. 3.5b - 3.5c)¹. The model can be divided in two main components: an encoding part and a

¹This model is aligned with AlphaStar (Vinyals et al., 2019) [117], RL neural model used by DeepMind for competing in the Starcraft strategic game. The agent uses several networks to embed the information from the game state (e.g., a Transformer network to represent the entities, a ResNet to embed the map, etc.), and multiple DNNs to compute the different actions. Although, in that case due to the game is partially observable an additional LSTM is also required to remember the sequence of previous states.

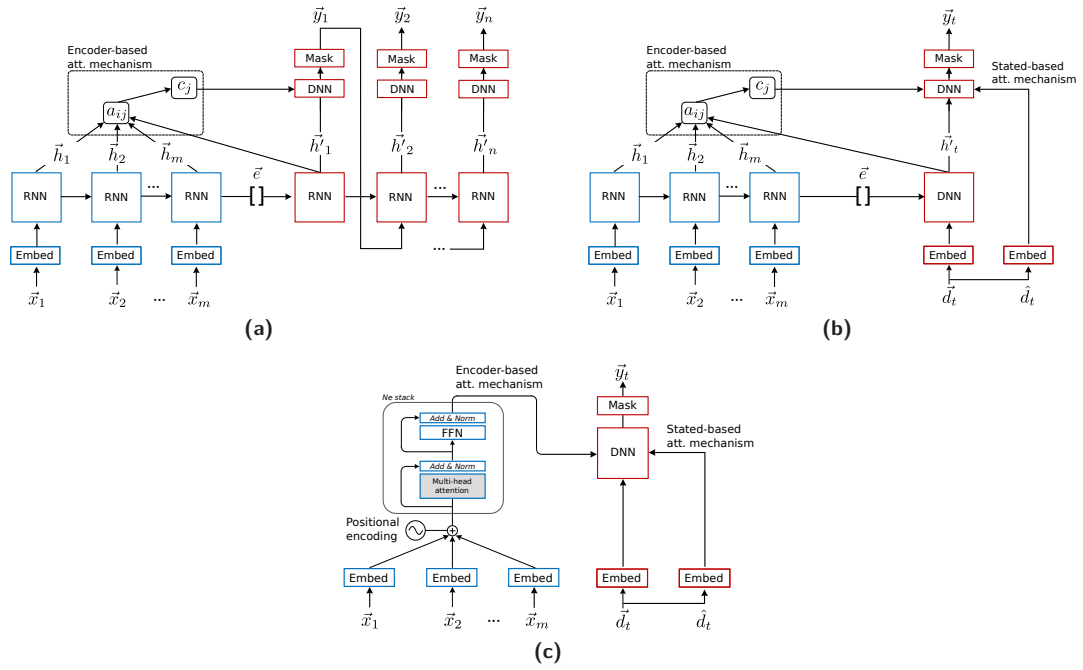


Fig. 3.5.: Architecture comparison between a: (a) traditional recurrent sequence-to-sequence model with an attention mechanism over the encoder and (b) & (c) our proposed iterative models. On (a), the hidden state of the decoder is commissioned to store the features of the partial solution. Whereas on (b) & (c), the model decodes each element on the solution fully observing the intermediate states on the problem. In particular, (b) preserves the recurrent encoder and (c) substitutes it in favor of a fully-attentional Transformer encoder.

DNN in charge of computing the output distribution. In Fig. 3.5b, the codification of the instance of the problem x is similarly done to previous model. It uses recursion to embed the definition of the problem instance into a single vector. Every element in the input x_j in recursively encoded, we refer to the final encoded vector as $\vec{e} = enc(\vec{x}_1, \vec{x}_2, \dots, \vec{x}_m)$. This corresponds to the static part of the input to our model, and does not change during the resolution. Then, the vector \vec{e} is combined with the state of the environment \vec{d}_t to create the state \vec{s}_t from which a DNN computes the policy distribution $\pi(a_t | s_t)$. This constitutes therefore, the dynamic part of the model, and it is evaluated in every interaction t until the whole solution is completed.

Alternatively, in Fig. 3.5c the input sequence is attended using a fully-attentional Transformer encoder. In this case, the decoder is able to focus on direct parts of the input sequence without using recursion. This model completely removes recursion from the solution.

3.4.2.1 Recurrent encoder-based attention mechanism

One side-effect of recurrent models is that performance comes down when there are complex relations between the sequences. To address this issue, usually attentional mechanisms are implemented. These mechanisms enable to attend on parts of the input sequence at the time, focusing just on the elements that provide relevant information to the decision. This technique was introduced in Natural Language Processing (NLP) to weigh the part of the sentence in which the translation must be focused. Examples of these variants of this encoder-based attention mechanism are (Bahdanau et al., 2014) [11], or (Luong et al., 2015) [70].

Attention mechanisms operate on recurrent models as follows. The input to the attention layer is set to the features of the encoder $\mathbf{h} = \{\vec{h}_1, \vec{h}_2, \dots, \vec{h}_m\} \in \mathbb{R}^F$, where m is the number of input

elements and F is the number of features in each node. The attention layer produces from them a new set of node features of potentially different cardinality, $f : \mathbb{R}^F \rightarrow \mathbb{R}^{F'}$. Also, features from the current state of the decoder \vec{h}'_j are extracted to create the attention layer. To transform these features into higher-level features, a linear transformation is used. To that end, a shared linear transformation, parametrized by the *weight matrices*, $W_1, W_2 \in \mathbb{R}^{F \times F'}$, is applied to every feature vector from the encoder element i and decoder step j . A self-attention then is performed on the nodes $v_a : \mathbb{R}^F \times \mathbb{R}^{F'} \rightarrow \mathbb{R}$ to compute the *attention coefficients*.

$$v_{ij} = v_a^\top \cdot \text{logit}(W_1 h_i + W_2 h'_j) \quad (3.3)$$

The resulting vectors v_{ij} , indicate the importance of the element j of the decoder, into the input element i . This formulation allows for each node in the model to attend on every other node, dropping all structural information. Finally, the softmax function is used to extract the weight of each element in the attention. This results in the *alignment scores*,

$$a_{ij} = \text{softmax}(v_{ij}) = \frac{\exp(v_{ij})}{\sum_k \exp(v_{ik})} \quad (3.4)$$

The final output of the attention mechanism is the *context vector* c_j , which is created as the weighed sum of the elements of the input sequence. This is the sum of hidden states of the encoder h_i weighted by alignment scores a_{ij} the model computes for each state depending on its importance. The context vector contain therefore the most relevant features at the decoding step. Formally, the context vector for the output step j is formally defined as:

$$c_j = \sum_i a_{ij} \cdot h_i \quad (3.5)$$

This formulation can be extrapolated from sequence-to-sequence models (Fig. 3.5a) to the Markovian model (Fig. 3.5b). To this end, it is only required to substitute the decoder step j with the iteration t on the resolution process to create the context vector c_t at that time-step.

3.4.2.2 Transformer encoder-based attention mechanism

Alternatively, the Transformer encoder can be detached from the Transformer network and be used to perform the encoding of the input sequence (see Fig. 3.5c). The Transformer encoder relies on stacked self-attention layers to achieve this task. Therefore, recursion is not required, and a direct attending over the input sequence is performed at every step.

Since the model does not present recurrence, in order to preserve the order in the sequences, a positional encoder [44] is used. The positional encoder utilizes sine and cosine functions of different frequencies to encode the relative distance between different input positions in the sequence. Thus, the resulting feature vector embeds its relative position in the sequence.

The Transformer encoder is formed by two layers. The first is a multi-headed self-attention mechanism, and the second a FFN layer. This pattern can be stacked to create deeper models, the number of stacks is denoted as N_e in Fig. 3.5c. The connections around these elements is done following a residual layout. This enables the model to skip on demand neural layers that do not add value in a specific instance.

In particular, the attention mechanism is a scaled dot-product attention. This consists in queries q , keys k and values v , where $q \in \mathbb{R}^{d_q}$, $k \in \mathbb{R}^{d_k}$ and $v \in \mathbb{R}^{d_v}$, that are operated to obtain the attention on the values. Specifically, the dot product of $q \cdot k$ is used to determine the alignment between the queries and the keys vectors. The result is weighed by the value v to create the attention vector. Formally each attention head i is defined as,

$$AttentionHead_i(q, k, v) = \text{softmax}(W_i^Q q \cdot W_i^K k^t) \cdot W_i^V v \quad (3.6)$$

where the parameter matrices are $W_i^Q \in \mathbb{R}^{d_{model} \times d_k}$, $W_i^K \in \mathbb{R}^{d_{model} \times d_k}$ and $W_i^V \in \mathbb{R}^{d_{model} \times d_v}$.

Instead of performing a single attention, this model proposes to linearly project the queries, keys and values several times, achieving different attention vectors and concatenating them to obtain the final representation. This multi-headed method allows to attend different representations at different locations. Formally,

$$AttentionMultihead = \text{concat}(Head_1, \dots, Head_h) \cdot W^O \quad (3.7)$$

where $W^O \in \mathbb{R}^{hd_v \times d_{model}}$ are the parameters of the output linear layer that operates over the h attention heads.

Relying on a Transformer encoder theoretically presents several advantages when compared to recurrent encoders. Here, distant items in the sequence can directly affect each other on the final representation. Also, the attention can be focused on discrete elements independent on its position on the sequence. However, the Transformer network also presents some drawbacks. The attention mechanism is dimensioned for the largest sequence, and a masking system invalids the excess of terms in case of input a shorter sequence. This provokes that the memory this model requires is large. Several alternatives have been proposed in this sense [32].

3.4.2.3 State-based attention mechanism

Defining the model as Markovian allows to introduce second attention mechanism. In this case, having access to state representation allows to build an attention mechanism over the state representation \vec{d}_t . The state-based attention mechanism extracts key features from the current state \hat{d}_t in the same way the attentional encoders do over the instance \mathbf{x} . The extracted information is used in combination with the context vector c_t as a *glimpse* mechanism [118] to introduce key information deeper into the model. Further information will be given in the particularized models.

3.4.3 Constraint management in NCO

As advanced in the introduction, currently NCO has been used in the literature to solve combinatorial problems without dealing with constraints. Existing works build specific models to ensure feasible solutions [118], or rely on masking schemes [83]. Nevertheless, this cannot be done for a large number of combinatorial problems. This Thesis proposes a strategy that allows to apply NCO to any constrained combinatorial problem. Specifically, we argue that actions that lead to an immediate violation of a constraint should be masked to directly to avoid exploring those infeasible solutions. And constraints that due to their nature cannot be verified before acting, and thus cannot be masked, are relaxed and incorporated into the objective function.

3.4.3.1 Limitations of Action-masked Networks

Current models used in NCO have limitations dealing with constraints. For example, Pointer Networks (PNs) [120] allow to solve problems that require to compute permutations over the inputs (e.g., the TSP and the Knapsack problem), however they are not directly applicable to other kind of combinatorial problems. Similar masking schemes over the output distribution can be applied to restrict the output space in other neural architectures. This is the case for example of [83], where for solving the VRP, the cities previously visited are masked to avoid selecting them later in the decision process.

Using masking schemes forces the neural model to produce solutions that are feasible and, therefore, ensures that the environment can evaluate the quality of the solution (produce a reward signal). Nevertheless, masking schemes cannot be applied to all constraint problems. Only in problems in which the validity of solution is verifiable during its construction, this technique can be used.

In general, combinatorial problems present constraint equations that only at the end of the episode, when the solution is obtained, their feasibility can be verified. This prevents the use of making schemes in many situations. To address this issue, we propose to define constrained problems as **Constrained Markov Decision Processes (CMDP)** (Altman et al., 1999) [5]. In this framework, the environment not only provides a reward signal but also **penalty signals generated from constraints dissatisfaction** (see Fig. 3.6). This way, constraints can be incorporated as penalty terms into the objective function and be evaluated along with the policy. Reward constrained policies guide therefore the agent to achieve feasible solutions.

3.4.3.2 Background on Constrained RL

Multiple approaches have been used in the literature for dealing with restrictions in RL problems. From a general prospective, different optimization criteria can be pursued: worst-case criteria, risk-sensitive criteria or a purely constrained criteria. A comprehensive overview of the research done in this area is given by (Garcia & Fernandez, 2015) [40].

In this sense, one has to assume whether during the learning process, constraints can or cannot be broken. Imagine that the agent is learning how to drive, one cannot allow to crash the car for exploring that this is action results in a bad outcome. Classical exploratory behaviours in RL assume that agent has to explore and learn different actions, and this is done ignoring the risky actions can potentially end in erroneous states. In principle, it is impossible to completely avoid these undesirable situations unless external knowledge is introduced. This can be done in two ways:

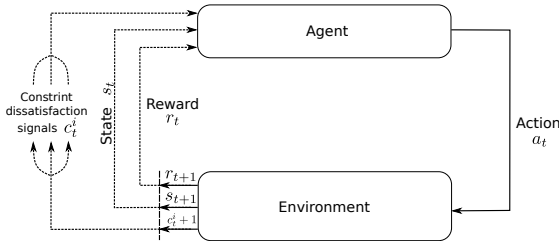


Fig. 3.6: Reinforcement Learning schema with constraint dissatisfaction indication. The agent iteratively interacts with an environment by carrying out an action A_t based on its observed state representation S_t at each step. In return it retrieves in addition to the reward signal R_{t+1} , signals that measure constraint dissatisfaction C_{t+1}^i . These signals enable the agent to measure the incorrectness of the actions and guide the agent towards feasible policies.

- **Forcing external knowledge:** one can incorporate external knowledge to provide initial procedure for the learning or derive a policy using a finite set of previously studied examples [102]. However, such initialization approaches are not sufficient to prevent dangerous situations which occur in later exploration.
- **Using Safety Layers:** additional layers that are added to the policy network to take possible unsafe actions predicted and output the closest actions that are safe. Is the case for example of [90, 33] for restricting continuous spaces.

Differently, in this work constraints can be breached during the learning, as the model is trained on a theoretical problem. Because of this, non-maskable constraints are treated as soft-constraints and incorporated into the objective policy function, this method is known as *Reward constrained policy optimization*. The main approaches to such problems are: (i) primal-dual methods [113, 23, 87], (ii) trust-region optimization [2], (iii) manual selection of the penalty coefficients [111].

3.4.3.3 Reward constrained policy optimization

Before introducing the Reward Constrained Policy Optimization (RCPO) method let us start defining the Constrained Markov Decision Process (CMDP) formulation used in this NCO approach. A CMDP augments an MDP with constraints that restrict the set of allowable policies for the model. Specifically, auxiliary cost signals are added $C_i : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow \mathbb{R}$ mapping transitions to cost, in the same way rewards do. These constraint dissatisfaction signals in the RCPO method are incorporated into the policy optimization function to prevent the model from these behaviours. That is, in the beginning the agent is free to explore *any behaviour*. As the learning advances the policy not only optimizes the objective but also the dissatisfaction of the constraints. This result in a policy that discourages constraints in expectation.

The RCPO method requires a parametrization of the policy, as it is over the objective expected reward function where the penalty is added. In particular, it resorts to Policy Gradients to learn the parameters of the stochastic policy $\pi_\theta(a_t|s_t)$ that, given as input the tuple composed by the instance of the problem and the state of the environment $s_t = \{\mathbf{x}, d_t\}$, assigns high probabilities to actions a_t that produce solutions with high reward, and low probabilities to those that do not. In Policy Gradients, the objective function $J_R^\pi(\theta)$ is defined as the expected reward for the policy π

$$J_R^\pi(\theta) \doteq \mathbb{E}_{\tau \sim \pi_\theta} [R(\tau)] \quad (3.8)$$

Non-maskable constraints are incorporate into Policy Gradients objective (3.8) using the Lagrange relaxation technique. This allows us to shape the objective function, proportionally penalizing

those policies that lead to infeasibilities. But first, for each constraint signal C_i , we define its expectation of dissatisfaction associated to the policy π as

$$J_{C_i}^\pi(\theta) \doteq \mathbb{E}_{\tau \sim \pi_\theta} [C_i(\tau)] \quad (3.9)$$

The primal problem becomes then to find the policy π^* that maximizes the expected reward subject to the satisfactions of the constraints,

$$\max_{\pi_\theta \sim \Pi} J_R^\pi(\theta) \quad \text{s.t.} \quad J_{C_i}^\pi \leq 0 \quad \forall i \quad (3.10)$$

In this expression Eq. (3.10), Π denotes the set of all stationary policies. The set of feasible policies for the CMDP defines therefore as, $\Pi_c \doteq \{\pi \sim \Pi : J_{C_i}(\pi) \leq 0\}$.

Using the Lagrange relaxation technique [20], the problem statement in Eq. (3.10) is reformulated as an unconstrained problem where the unfeasible solutions are penalized. The result is a multi-objective function defined as

$$\begin{aligned} g(\lambda) &= \max_{\theta} J_L^\pi(\lambda, \theta) = \max_{\theta} [J_R^\pi(\theta) - \sum_i \lambda_i \cdot J_{C_i}^\pi(\theta)] = \\ &= \max_{\theta} [J_R^\pi(\theta) - J_\xi^\pi(\theta)] \end{aligned} \quad (3.11)$$

where $J_L^\pi(\lambda, \theta)$ denotes the Lagrangian objective function, $g(\lambda)$ stands for the Lagrange dual function, and $\lambda \in \mathbb{R}^+$ are the Lagrange multipliers, i.e., the penalty coefficients. Also, in this equation (3.11) the term $J_\xi^\pi(\theta)$ is introduced to define the expected penalty, which is computed as the weighted sum of all expectation of constraint dissatisfaction signals.

It is noteworthy that setting the Lagrange coefficients λ_i is a multi-objective problem where for each set of penalty coefficients there exists a different optimal solution, also known as Pareto optimality [115]. Balance the different coefficients is challenging, selecting the coefficients is hard as the effect of their values on the overall policy is not straightforward. In practice, the coefficients are selected through a time consuming hyper-parameter tuning. Moreover, the interference between the constraints into the objective function may lead to additional plateaus, as they compete over the policy [97]. This makes even harder and domain dependent its selection. In this sense, we use **prior knowledge on the problem to perform a manual selection of the penalty coefficients**, although the optimal value can also be automatically discovered using other techniques. E.g., using the Primal-dual algorithms though a multi-timescale learning [113, 21]. In these algorithms the policy update is on a faster time-scale than the multiplier update. Thus, effectively, this approach iteratively converge to solutions as if the dual problem was solved.

The gradient of the Lagrangian objective function $J_L^\pi(\theta)$ is derived using the log-likelihood method. This derivation process is similar as deriving the expected reward, method introduced in [122]. The resulting gradient equation is

$$\nabla_{\theta} J_L^\pi(\theta) = \mathbb{E}_{\tau \sim \pi_\theta(\cdot|s)} \left[\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(S_t, A_t) \cdot L(S_t, A_t) \right] \quad (3.12)$$

where

$$\begin{aligned} L(a|s) &= R(a|s) - \xi(a|s) = \\ &= R(a|s) - \sum_i \lambda_i \cdot C_i(a|s) \end{aligned}$$

$L(a|s)$ denotes the penalized reward obtained in each iteration, calculated by subtracting to the reward signal $R(a|s)$ the weighed sum of all the constrained dissatisfaction signals $C(a|s)$.

Remark 2: Dealing with constraints as penalties is key point in highly constrained environments. Providing bad rewards to unfeasible solutions can flatten the objective function, which leads to a lack of information to infer a competitive policy. Without this relaxation technique, it would be near impossible for the agent to achieve a feasible region, as it would not experience enough positive rewards. The problem of sparse reward is well known in RL [56].

Lastly, the expectation is approximated using Monte-Carlo sampling, where B problem instances are drawn from the problem distribution $x_0, x_1, \dots, x_{B-1} \sim \mathcal{X}$. This results in the fundamental gradient equation used in this work.

$$\nabla_{\theta} J_L^{\pi}(\theta) \approx \frac{1}{B} \sum_{b=0}^{B-1} \cdot \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(S_t^b, A_t^b) \cdot L(S_t^b, A_t^b) \quad (3.13)$$

Intuition on the selection of the Lagrange coefficients

In the following text some intuitions on the selection of the Lagrange multipliers are provided. Let us refer to the optimal of the primal problem described in Eq. (3.10) as P^* , the maximum value that the objective function can obtain subject to the satisfaction of the constraints. Using the Lagrange relaxation technique, this problem is transformed into an unconstrained problem where unfeasible solutions are penalized, Eq. (3.11). The resulting dual function $g(\lambda)$ is the point-wise maximum of the primal with respect to the policy.

The dual function provides an upper bound on the value of the primal, i.e., $g(\lambda) \geq P^* \quad \forall \lambda \in \mathbb{R}^+$. The tighter the bound, the closer the policy obtained is to the optimal solution of the primal problem. Hence, the dual problem is that of finding the best bound:

$$D^* \doteq \min_{\lambda \in \mathbb{R}^+} g(\lambda) \quad (3.14)$$

The dual function is always convex, even though the primal function and the constraints are non-convex. Due to this property, and the peculiarities of problem addressed, some conclusions can be extracted. Generally, all the constraint signals C_i are positive definite, in other words, a feasible solution would be the one in which all those expectation of constraint dissatisfaction signals were equal to zero. Nevertheless, there might not exist in the whole space of weights θ a configuration of the neural network that fulfills this requirement. If it exists, the problem has **zero duality gap**, $\Delta = D^* - P^* = 0$. Strong duality holds because there exists a point that deletes the penalty term (complementary slackness). Solving the primal problem and the dual problem would be essentially equivalent.

In general, constraints are not positive definite and thus provide a subsidy when are satisfied that shapes the dual as depicted in Fig. 3.7a. In that case, there exists an optimal coefficient λ^* that provides the best bound.

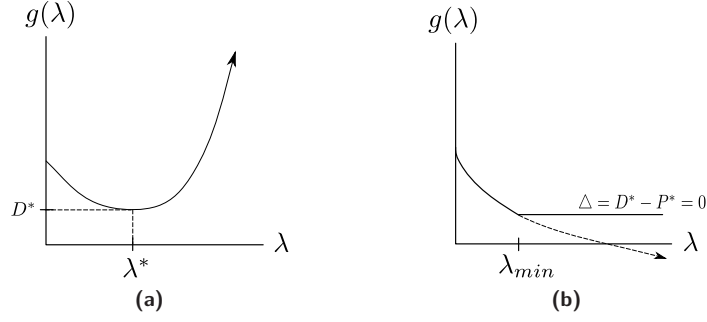


Fig. 3.7.: General dual function representation in cases where: **(a)** a benefit in the feasible regions exists, **(b)** no subsidy is received in the feasible region or a feasible region does not exist.

However, in the scenario here described where there is no subsidy due to the constraints, two cases occur depending whether a feasible point is proven to exist or not (Fig. 3.7b). In the case it exists, there will be a λ_{min} at which point we can guarantee that the penalization is large enough to affirm that the optima of the dual function D^* corresponds to the optima of the primal problem P^* . This value is a priory unknown but can be estimated. It might seem that all values for λ between λ_{min} and ∞ are equally valid, but we will clarify this later.

In the case that there is no feasible point for any policy $\pi_\theta \notin \Pi_c$, there is also no primal optima P^* . The dual function constantly decreases with λ , as the function is more and more penalized. The goal is then to find the Lagrange coefficients λ that establish the desired commitment between the expectation of a penalty J_C^π and the expectation of reward J_E^π . To clarify this concept, let us analyze the extreme values for λ . If $\lambda = 0$, then the Lagrangian $J_L^\pi = J_E^\pi$, the optimizer is going to minimize the expected power consumption without paying attention to the constraints. Similarly, if $\lambda \rightarrow \infty$, by the law of big numbers, the Lagrangian is going to converge to the penalization function J_C^π . Therefore, the policy that the agent is going to infer is the one that achieves by mean fewer misstatements without taking into account the optimality of the solution. In this case, there is no λ that maximizes the dual function $g(\lambda)$ to give us a reference point on what penalization is enough in order to the feasible region, or in this case, the region that presents a minimum in constraint dissatisfaction, surpassing the rest of the function.

In practice, if the neural network is a good enough approximator, then a weight configuration exists in which $J_C^\pi \approx 0$. This gives us an intuition that the region to find the Lagrange coefficients is λ_{min}^+ . Once the penalty coefficients are set, we obtain the Lagrangian function $J_L^\pi(\theta)$ to optimize. This Lagrangian is a non-convex function and, therefore, it cannot be guaranteed that the optimizer is going to achieve the global optima. Normally this function is optimized until a saddle point or a local optima is reached. A priori, we do not have information on the convergence point and a fine tune of the λ is going to be needed to set the desired commitment between optimality and constraint dissatisfaction. Notice that during this fine tuning we move by following the trajectory that joins the local optima or saddle point achieved during the learning in the extreme cases ($\lambda = 0$ and $\lambda = \infty$). In theory, we should experience benefits in favor of the goal we want to achieve by controlling the λ . Yet, in reality, nothing guarantees that during this procedure a different optimization path with a better or worse performance is achieved.

3.4.4 Optimization algorithms

Finally, this Chapter ends outlining the optimization methods that would be required to optimize the cost function. In this sense, some key aspects are provided on the selected alternatives in order to align them with the proposal. As mentioned, this Thesis is focused in gradient-based methods. Although there exists other derivative free optimization and evolutionary alternatives (e.g., CEM or CMA-ES) that have shown to almost match in performance to PG methods in some scenarios [95], **gradient-based methods are considered the best option to optimizing deep neural models**, not only due to the results but also because of their facility to tune competitive algorithms.

Stochastic gradient descent (SGD) is the simplest way to maximize the objective function pursued in policy optimization $J(\theta)$. Namely, to find the parameters of the neural network $\theta \in \mathbb{R}^d$, that achieves the best performance, SGD computes the gradient of the cost function w.r.t. to the parameters θ ,

$$\theta \leftarrow \theta + \alpha_t \nabla_{\theta} J(\theta)$$

Robbins-Monro [93] proved the converge of this expression under some requirements. These are related with the value of the step-sizes α_t :

$$\sum_{t=1}^{\infty} \alpha_t = \infty; \quad \sum_{t=1}^{\infty} \alpha_t^2 < \infty$$

E.g., a particular sequence of steps which satisfy these conditions, have the form $\alpha_t = \alpha/t$. However, in practice a small enough α that does not necessarily decrease during the optimization process is an equally good option.

However, SGD presents the some challenges:

- Choosing a proper learning rate is difficult. A learning rate that is too small leads to slow convergence, while a learning rate that is too large can hinder convergence and cause the loss function to fluctuate around the minimum or even to diverge.
- Additionally, the same learning rate applies to all parameter updates. If the data is sparse and the features have very different frequencies, one might not want to update all of them to the same extent, but perform a larger update for rarely occurring features.
- Another key challenge of minimizing highly non-convex error functions common for neural networks is avoiding plateaus, which makes it notoriously hard for SGD to escape as the gradient is close to zero in all dimensions.

Gradient descent is a first-order optimization method. It only takes the first order derivatives of the loss function into account and not the higher ones. This means that it has no clue about the curvature of the loss function. It can tell whether the loss is declining and how fast, but cannot differentiate between curves. For this reason, it presents problems navigating where there is a different second derivative for each direction at a point. The condition number of the Hessian at a point measures how much the second derivatives differ from each other. When the Hessian has a poor condition number, gradient performs poorly. This is because in one direction the gradient increases rapidly, while in another direction does slowly. In these scenarios, SGD oscillates across the slopes only making negligible progress along one direction towards the local optimum. This issue can be resolved using information from the Hessian matrix to guide the

search. Using second-order curvature information to find search directions can help with more robust convergence for non-convex optimization problems. An alternative for that is the Newton's method.

Second-order methods such as Newton's method can give an ideal step size to move in the direction of the gradient. Since they have information about the curvature of our loss surface. Newton's Method does it by computing the Hessian matrix, which is a matrix of the second derivatives of the loss function with respect of all combinations of the weights. The Hessian gives us an estimate of the curvature of loss surface at a point. However, computing the Hessian requires to compute gradients of the loss function with respect to every combination of weights, which is of the order of the square of the number of weights present in the neural network. Though it is time-consuming and no practical to calculate.

As an alternative, quasi-Newton methods gradually build up an approximate Hessian matrix by using the gradient information from previous iterations. Quasi-Newton methods combine the speed of Newton's algorithm with the scalability of first-order methods by incorporating the curvature information. Two of examples of these methods are the Broyden-Fletcher-Goldfarb-Shanno (BFGS) or Symmetric Rank 1 (SR1). These methods have shown good performance in supervised learning [18]. Although there is a discussion in the deep learning community on whether these algorithms represent the best alternative. Currently, momentum accelerated first-order methods are widely used due to the effectiveness they have shown on a wide variety of environments. In general, momentum-based optimizers are seen as the primary option to deal with these kind of problems [50].

3.4.4.1 Momentum-based optimizers

As mentioned, a good alternative to improve SGD and accelerate its convergence resides in including momentum. These momentum-base first-order methods use heuristics to guide the search based upon the past behavior of gradient. Some of the most relevant implementations in the community are:

- **Momentum** [91]: algorithm that accelerates SGD in the relevant direction and dampens oscillations. This method adds momentum including a fraction of the update vector of the past time step to the current update vector. The update is performed as follows,

$$\begin{aligned} v_t &= \gamma v_{t-1} + \alpha \nabla_{\theta} J(\theta) \\ \theta &\leftarrow \theta + v_t \end{aligned} \tag{3.15}$$

where the γ term weights the momentum. It increases the step for dimensions whose gradients point in the same directions and reduces updates for dimensions whose gradients change directions.

- **Nesterov Accelerated Gradient** [84]: this method includes an approximation of the next position of the parameters. This allows to effectively look ahead by calculating the gradient not w.r.t. to our current parameters θ but w.r.t. the approximate future position of them:

$$\begin{aligned} v_t &= \gamma v_{t-1} + \alpha \nabla_{\theta} J(\theta + \gamma v_{t-1}) \\ \theta &\leftarrow \theta + v_t \end{aligned} \tag{3.16}$$

Nesterov accelerated gradient first makes a big jump in the direction of the previous accumulated gradient, measures the gradient and then makes a correction. This anticipatory update prevents it from a too fast approximation and results in increased responsiveness, which has significantly increased the performance, specifically in recurrent models [16].

Momentum accelerated methods suffer from a well-known problem. In case the momentum acquired is too large, they are likely to miss the local minima. As an alternative, adaptive learning rate can be applied to momentum methods to modify the learning rate to the parameters, performing smaller updates (i.e. low learning rates) for parameters associated with frequently occurring features, and larger updates (i.e. high learning rates) for parameters associated with infrequent features. For this reason, it is well-suited for dealing with sparse data. An adaptive learning rate is used in the following methods:

- **Adaptive Gradients (AdaGrad)**: provides a simple approach for changing the learning rate over time. This method uses a different learning rate for every parameter θ_i at every time step t . To this end, the learning rate is adjusted dividing it by the sum of the squares of the gradients w.r.t. θ_i obtained up to the time step t .

Adagrad's main weakness is that this method monotonically decreases the learning rate. Since the learning rate is divided by a positive increasing term, the accumulated gradients, it keeps diminishing during the training. This causes the learning rate to shrink and eventually become infinitesimally small, at which point the algorithm is no longer able to acquire additional knowledge.

- **Root Mean Squared Propagation (RMSprop)**: very close to Adagrad, except for the fact that it does not retain the sum of the gradients, but instead computes an exponentially decaying average. RMSprop instead of inefficiently storing the previous squared gradients, it computes a decaying average of all past squared gradients. The running average at time step t then depends only on the previous average of the current gradient.
- **Adaptive Moment Estimation (Adam)** [63]: this adaptive moment estimation method is simply a combination of Momentum and RMSProp. Adam in addition to computing the exponentially decaying average of past squared gradients like RMSprop, it also keeps an exponentially decaying average over the past gradients, similar to Momentum.

In general, adaptive momentum gradient descent optimization methods outperform their counterparts in a wide variety of environments. However, they are highly sensitive to hyperparameter configuration.

3.5 Conclusions

This Chapter argues that CO problems present the Markov property, and so, are suitable to be addressed using traditional RL techniques. This approach benefits the results as accessing the fully observable state of the environment is more reliable than doing on memories as previous models do. In this sense, two different neural agent are proposed. Both models encode the instance of the problem, extract the features from the definition and then incorporate the state of the environment in order to perform an iterative resolution process in which the solution is constructed based on the action the model performs at each time-step. Particularly in the first case, a RNN is used to encode the input sequence that defines the problem description and in the second case, a Transformer encoder is used to build a fully-attentional model that removes recursion from the solution. This is aligned with the trend in the RL community which affirms

that establishing non-direct relations in the agent (as occurs with recursion) hampers the learning process.

Also, a methodology for handling constraints is introduced here. To this end, the **Constrained Markov Decision Process (CMDP)** formulation of the model is used to incorporate penalty signals into the model indicating the constraint dissatisfaction levels. We conclude that actions that lead to an immediate violation of a constraint should be masked to directly avoid exploring those infeasible solutions. However, due to their nature, not all constraints can be verified before acting, and thus, cannot be masked. To that end, we propose **dealing with non-maskable constraints by incorporating them into the Policy Optimization objective via Lagrange multipliers**. Accordingly, some intuitions in the selection of the penalty coefficients are provided.

Finally, this Chapter ends outlining that momentum based optimizers are the best option for optimizing NCO models. An extensive literature has been revised including first order, second order and quasi-Newton methods to conclude that first order with momentum, and particularly, adaptive momentum alternatives are the best suitable option in these high-variance environments.

Proposal for a model building process and experimentation

“If a machine is expected to be infallible, it cannot also be intelligent.”

— Alan Turing

Contents

4.1	Motivation for studying the learning process in a toy CO problem	44
4.2	Methodology for improving the learning process	45
4.2.1	Convergence analysis	45
4.2.2	Enhance exploration strategies in deep RL	46
4.2.3	Monotonic improvements with Trust-region optimization	52
4.2.4	Improving the model with a self-competing strategy	60
4.2.5	Search strategies	62
4.3	Proposed framework for addressing NCO	63
4.4	Experimentation on the Job-shop Scheduling Problem	64
4.4.1	Job-shop Scheduling Problem with limited idle time	64
4.4.2	Particularized models	64
4.4.3	Learning algorithm: PPO with self-competing baseline	66
4.4.4	Results on the Job Shop Problem	66
4.5	Conclusions	70

So far in this Thesis, the baseline framework for building a Markovian NCO model has been discussed. This includes the different learning algorithms (Sec. 3.4.1), the neural model that constitutes the agent (Sec. 3.4.2), the methodology for addressing constraints (Sec. 3.4.3) and the optimization algorithms (Sec. 3.4.4). In this Chapter, these ideas are all put together to construct a proof-of-concept from where to draw an analysis. To this end, a toy combinatorial problem is used for testing the different alternatives. In this scenario, optimal solutions can be computed, and therefore a performance comparison against the optima can be obtained. But also, the learning procedure is much faster. During the analysis several experiments are drawn to tune the different alternatives. Relying on a reduce model significantly decreases the learning time (which can be very time-consuming) and enables to perform broader comparisons to lead better conclusions. The outcome extracted here would be of utmost interest to extrapolate behaviours applicable to further models.

Finally, a complete experimentation is performed in a well-known combinatorial problem, the Job-shop Scheduling Problem (JSP). This problem has been largely studied in the literature and there exists an extensive background of algorithms to solve it. In particular, the experimentation is done for different orders of the problem, comparing the solutions against well-documented heuristics, metaheuristics and constraint solvers. As argued, the presented RL model produces

fast intuitions on the problem, therefore note that a fair comparison can only be done against greedy heuristics. Although other resolution methods are also included in the experimentation to have a wider view on the results.

4.1 Motivation for studying the learning process in a toy CO problem

In this section, experiments to study the learning process on constrained combinatorial problems are conducted. To this end, a toy CO problem is used to draw general conclusions on the different learning approaches. Specifically, a small size Bin-packing problem is used to this end. In order to implement this neural model, tensor oriented libraries are required (e.g., Tensorflow [1] or PyTorch [86]). Neural networks are function approximators composed of differentiable operations. Hence, to train these models, automatic differentiation libraries that efficiently compute the backpropagation algorithm are required. The resulting learning process is depicted in Fig. 4.1. Here, for each iteration, the approximation obtained on the batch for the expected reward J_R^π , baseline estimator b , penalization J_ξ^π , and Lagrangian J_L^π are depicted. As shown in this example, at the beginning of the learning process, the agent generates random solutions, obtaining high penalty for those actions. Therefore, at the start, the agent only focuses on constraint satisfaction and ignores the underlying objective of the problem. As the learning progresses, the agent improves its policy correcting its weights θ via gradient descent to minimize the Lagrangian objective function (Eq. 3.13). This process continues until the model converges. At that point, the performance of the model does not improve no-matter how long the learning process is extended.

As mentioned, this analysis is done in a small toy combinatorial problem in which it is viable to compute for all instances the optimal solution. In the particular case exposed, the objective function is set following the indications on the selection on the Lagrange multipliers stated in Section 3.4.3.3, and we verify that the penalty coefficients are high enough to not participate on the optimal solution. Under this circumstances, the optimal expected penalty $J_\xi^{\pi^*} = 0$. In that case, and as Fig. 4.1 depicts, the model achieves close to optimal results, but the model still presents an optimality gap that remains no matter for how long the model is trained.

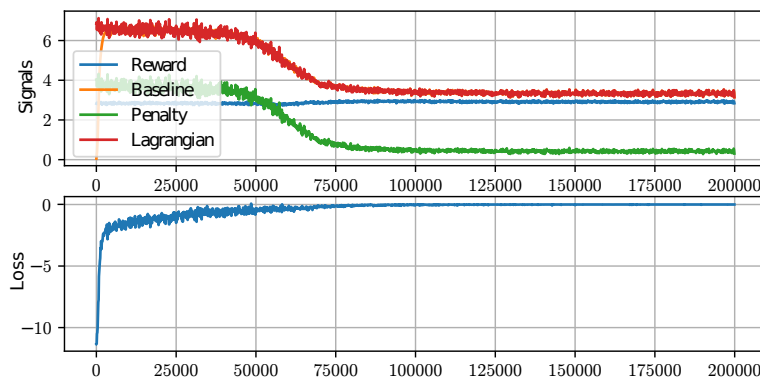


Fig. 4.1.: Learning process on a toy combinatorial optimization problem; the immediate reward, penalty, Lagrangian and baseline estimator are depicted. Below, the convergence of the objective function is shown. This objective function maximizes to zero when the policy term becomes deterministic, at that point the policy stays constant and the learning process is concluded.

Even though this toy problem represents a small environment in which all the optimal solutions are explored for all instances. The model does not infer a policy to optimally select the best solution for each instance. Note that exploring the optimal solution does not mean to learn it. These systems do not rely on memory, a positive exploration is only used to correct the policy π_θ in the right direction. One might think that is due to the non-linear approximator used to infer the policy (the neural agent) is not complex enough to infer the optimal behaviour. In this scenario, the neural network has to embed for all instances of the problem the right solution. However, this is disproved. After computing the optimal labels and training the model performing a supervised learning procedure, the agent is capable of overfitting the whole space of solutions. Then, we conclude that in this case the neural agent does not represent an obstacle to achieve the optimal behavior, but the RL learning algorithm limits the policy achieved.

This experiment lead us to the next question, what enhance learning methods could be applied to improve the results achieved, and therefore reduce the optimality gap? And ultimately, it is possible using RL to learn the optimal policy π^* even for simple CO problems with a small combinatorial space? These questions will be answered during the analysis performed in the following sub-sections.

4.2 Methodology for improving the learning process

In order to improve the results achieved in the previous experiment, and therefore reduce the optimality gap, several techniques can be applied. In this sense, in the following sub-sections these strategies are evaluated:

- perform a convergence analysis,
- enhance exploration with entropy regularization,
- reduce policy degradation with trust-region optimization,
- and evaluate search strategies at inference.

4.2.1 Convergence analysis

One of the key aspects to analyze in policy optimization methods is the convergence. In policy optimization methods, a loss function $J(\theta)$ is defined and numerical optimization is used to find the satisfactory parameters θ that optimize the model. Policy optimization methods present local convergence and the challenge of minimizing these highly non-convex functions is avoiding getting trapped in their numerous local suboptima. To this end, knowing if the learning mechanism converges to a local minimum or stays in a saddle point during the optimization process is key to apply different optimization techniques in the model.

To find information about the convergence of the model, the second derivative is used. One can think of the second derivative as a measure of the curvature. When functions have multiple input dimensions, there are many second derivatives, which are collected together into a matrix called the Hessian matrix H . Anywhere that the second partial derivatives are continuous, implies that the Hessian matrix is symmetric at that point. Because of the Hessian matrix is real and symmetric, it can be decomposed it into a set of real eigenvalues and an orthogonal basis of eigenvectors. The second derivative in a specific direction represented by the vector d is therefore given by $d^\top H d$. When d is an eigenvector of the Hessian H , the second derivative in that direction is given by the corresponding eigenvalue.

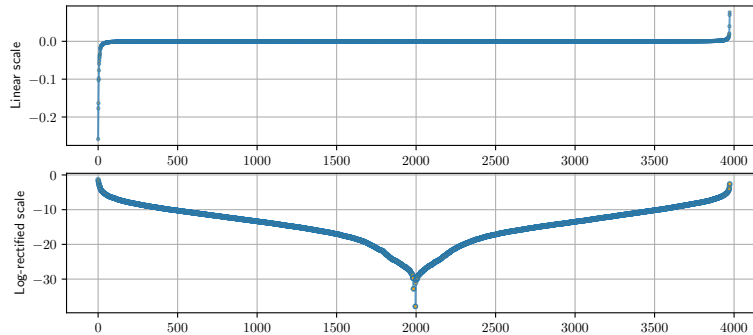


Fig. 4.2.: Eigenvalues of the Hessian of the policy network π_θ . The eigenvalues are presented in a linear scale and also in a rectified logarithmic scale. Results consistently show both positive and negative eigenvalues on trained models.

Numerical optimization theory state that in these multidimensional functions the eigendecomposition of the Hessian matrix can be used to determine whether a critical point is a local maximum, a local minimum, or a saddle point. At that critical point where $\nabla_\theta J(\theta) = 0$, the following deductions can be done:

- If the Hessian is positive definite (e.g., all eigenvalues are positive) at the critical point, then it is a local minimum of function.
- Similarly, if the Hessian is negative definite (e.g., if all eigenvalues are negative), then this point is a local maximum of function.
- If the Hessian has positive and negative eigenvalues, it is a saddle point of function.
- If any of the eigenvalues are zero, the test is inconclusive. This is because the univariate second derivative in the cross section corresponding to the zero eigenvalue is inconclusive.

This analysis has previously arisen in the literature [34]. By and large, there is a consensus that in general, deep models do not achieve a local minima but saddle points. In this work, this is empirically verified. Even relying on a small neural network to address this toy CO problem, and the learning process is perpetuated until the gradient vanishes, the eigenvalues of the Hessian do not achieve positive values for all directions (depicted in Fig. 4.2). Instead, **plateaus and ultimately saddle points are more likely to occur**. For this reason, and to avoid from getting stuck on these flat regions, momentum accelerated optimizers are key to improve the training on these models.

4.2.2 Enhance exploration strategies in deep RL

Another technique that can be applied to improve the results is enhancing the exploration. Several classic exploration techniques are embedded into RL algorithms (e.g., epsilon-greedy, upper confidence bounds, Boltzmann exploration, Thompson sampling). In this Section, enhance exploration strategies that can be applied specifically in deep RL models are considered. The following alternatives are evaluated when neural networks are used for function approximation:

- **Entropy Regularization:** this method adds an entropy term into the loss function, encouraging the policy to preserve stochasticity.
- **Noise-based Exploration:** alternatively this method enhances exploration adding noise into the observation, action or even the state space.

- **Intrinsic Rewards:** in addition to the previous approaches and with a particular interest for solving hard-exploration problems is to augment the environment reward with an additional bonus signal to encourage exploration.

In this Thesis, the Entropy Regularization technique and Intrinsic exploration bonuses are evaluated on the scope of CO problems.

4.2.2.1 Entropy Regularization

As mentioned, **Entropy Regularization** [4, 129] can be applied to improve the results in policy optimization methods. One might wrongly think that the longer the model is trained the better is the performance. However, policy optimization methods (e.g., PG or AC) rely on a "learning window" to define their policy. These methods begin the learning process with a stochastic policy, which is used to perform exploration (Annex B.4), and it is during the learning process that the policy converges to a deterministic one. At that point, the learning window closes and the performance of the model stays constant no matter for how long the learning procedure continues. Hence, if during the learning process the model does not sample a large number of diverse state-action pairs, it may converge to a poor policy.

A technique to extend the learning window, and therefore catch more information from the problem is Entropy Regularization. This technique prevents policies from becoming deterministic too quickly encouraging the selection of stochastic policies. In this framework, a new entropy-augmented objective is optimized $J^\beta(\theta)$, which is more preferred with stochastic behaviours. To achieve this goal, along with the environment reward, the agent gets a bonus proportional to the entropy of the policy at each time-step. The RL problem under this description stands to optimize the following equation

$$J^\beta(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} [R_{t+1}^\beta] = \mathbb{E}_{\tau \sim \pi_\theta} [R_{t+1} + \beta \mathbb{H}_\pi(S_t = s)] \quad (4.1)$$

being the entropy of the policy \mathbb{H}_π obtained as

$$\mathbb{H}_{\pi_\theta}(S_t = s) \doteq \mathbb{E}_{a \sim \pi(\cdot|s)} [\log \pi_\theta(A_t|S_t)]$$

The gradient of the entropy-augmented objective is calculated as follows,

$$\begin{aligned} \nabla_\theta J^\beta(\theta) &= \nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta} [R^\beta(\tau)] = \\ &= \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(A_t|S_t) \cdot Q_{\pi_\theta}^\beta(S_t, A_t) + \beta \nabla_\theta \mathbb{H}_{\pi_\theta}(S_t) \right] \\ &\approx \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(A_t|S_t) \cdot Q_{\pi_\theta}(S_t, A_t) + \beta \nabla_\theta \mathbb{H}_{\pi_\theta}(S_t) \right] \end{aligned} \quad (4.2)$$

where $Q_{\pi_\theta}^\beta$ is the expected discounted sum of entropy-augmented rewards. Which for the purpose of the method can be approximated by the action-value function.

In this equation, the hyperparameter β regulates the trade-off between the importance of the entropy term against the environment's reward. When β is high, large entropy levels are added to the policy, causing the policy to be more stochastic; on the contrary, with low values of β , the policy is preferred with deterministic behaviours. The entropy term can be added directly to the

loss function weighed with the β parameter or it can be included shaped by a decay function $\psi^\beta(t)$. In the first case the entropy term is present during the whole learning process, which prevents the policy from being completely deterministic. Alternatively, the second option uses a decay function to ensure that the entropy term vanishes and ultimately the policy becomes deterministic.

The shape of the decay function is relevant for the performance of the model. Its selection indicates the way in that the entropy term impacts during the learning process, and ultimately determines for how long the learning window stays opened. In the experimentation the following entropy augmented objective has been tested,

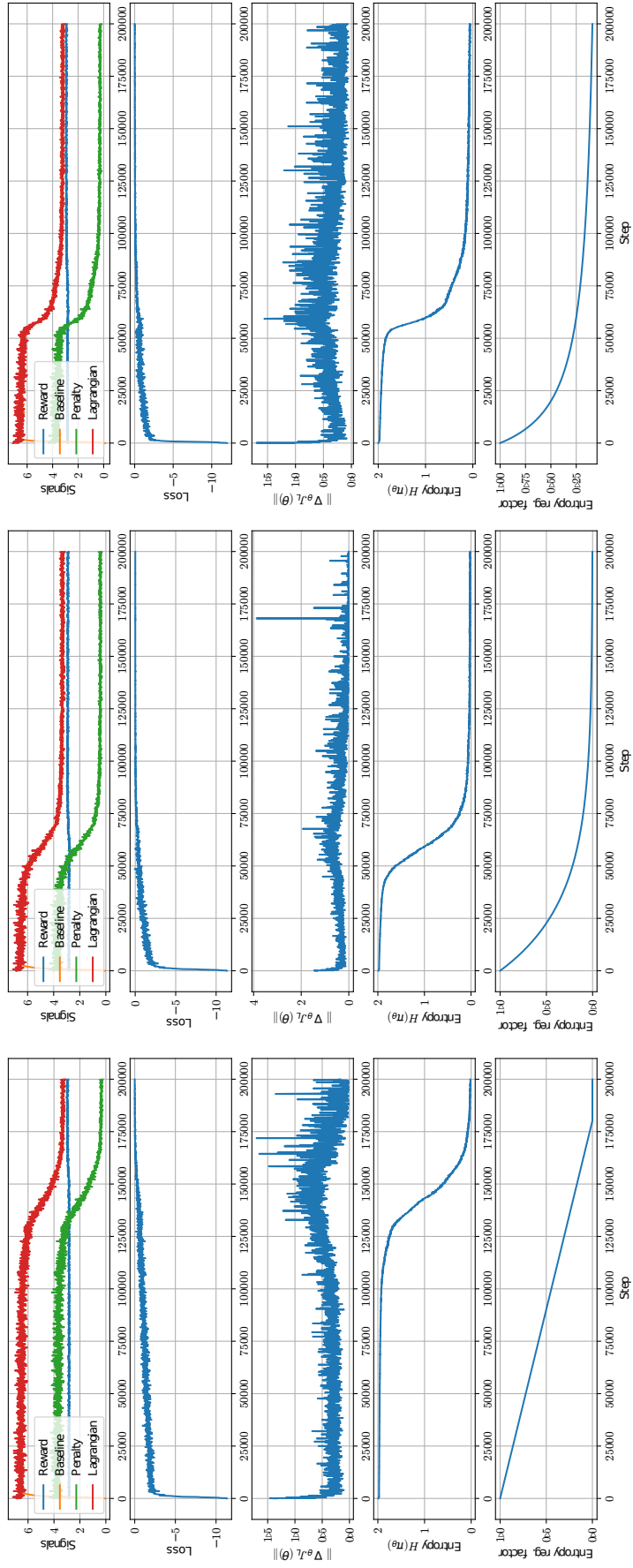
$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(A_t|S_t) \cdot Q_{\pi_\theta}(S_t, A_t) + \psi^\beta(t) \cdot \nabla_\theta \mathbb{H}_{\pi_\theta}(S_t = s) \right] \quad (4.3)$$

with the following decay functions $\psi^\beta(t)$ to shape the decay of the entropy term \mathbb{H}_{π_θ} :

- (a) Linear decay
- (b) Exponential decay
- (c) Inverse decay

The results of the analysis are depicted in Fig. 4.3. As can be observed, the different decay functions (a)(b)(c), produce **different convergence patterns**, this is captured in the entropy term. We conclude from the experiment that the policy convergence when the entropy factor decreases below a threshold, in the example this occurs for a value of $\psi^\beta(t) \approx 0.3$. Before getting this value the entropy added to the system is excessive, which prevents the model from learning good behaviors. Once the entropy function decreases below this threshold, the agent is able to infer some positive policies and begins to converge towards them.

As mentioned, the aim of this method is to preserve during the convergence process higher levels of exploration than policy optimization naturally would. In this sense, the longer acceptable levels of entropy are maintained, the better is the exploration for the final outcome. Particularly, from this experiment we observe that the **regularization effect experienced is more prominent using an inverse decay function (c)**, as this function rapidly declines surpassing the threshold but then it maintains for a long period suitable levels of entropy. Finally, the effect of entropy regularization can be also observed in the norm of the gradient of the loss function $\|\nabla_\theta J(\theta)\|$. This function acquires large values during the convergence process indicating that the model is enhancing exploratory behaviors.



(a) Linear decay (b) Exponential decay (c) Inverse decay

Fig. 4.3.: Comparison on the learning performance of different entropy regularization decay functions: (a) Linear decay, (b) Exponential decay and (c) Inverse decay.

4.2.2.2 Intrinsic Rewards as Exploration Bonuses

The exploratory behaviour of the model can be also increased using exploration bonuses. Traditional RL algorithms perform well in scenarios with dense rewards. However, they struggle with sparse rewards or for the interest of this Thesis, when we look for the optimal solution in high combinatorial spaces. To overcome these difficulties, Intrinsic Rewards build an auto-regulated mechanism that encourages the agent to explore novel states without any external supervision, just curiosity. To promote exploration this technique gives additional rewards on novel strategies. In this framework, the reward is formed of two terms: an **extrinsic reward** r^e obtained directly from the environment and a synthetic **intrinsic reward** r^i built to encourage exploration. Formally, $r_t = r_t^e + \eta r_t^i$, where η is a hyperparameter adjusting the balance between exploitation and exploration.

Most formulations of Intrinsic Rewards techniques fall into two main categories: 1) directly encourage the agent to explore novel states or, 2) encourage the agent to reduce the error predicting the consequences of its actions (i.e., its knowledge about the environment). In the first case the model counts how many times a state has been encountered and assigns a bonus accordingly. This bonus guides the agent's behavior to prefer rarely visited states over common states. This is known as the Count-based Exploration method. However, this method does not scale well. A different approach to make this strategy possible on high-dimensional spaces is to map states into hash codes so that states become trackable [112]. Alternatively, in the second scenario, intrinsic exploration bonuses are rewarded to improve the agent's knowledge about the environment. The agent's familiarity with the environment dynamics is estimated through a prediction model. In such case, curiosity is defined as the error in the ability the agent presents to predict the consequences of its own behaviour. In other words, the more novel a state is (measured using the prediction capabilities of the agent), the bigger the error to predict the state would present. This error is used to derive an intrinsic reward that motivates the agent to keep exploration. This self curiosity-driven RL strategy is known as **Prediction-based Exploration**, and it has achieved great success in otherwise unapproachable problems.

Several techniques can be applied to predict the model dynamics. This work, distinguish two main approaches: approximating the forward dynamics or the inverse dynamics.

Forward Dynamics

Learning a predictor of the forward dynamics prediction model is the most common way to approximate how much knowledge the model obtains about the environment. This method captures the capability the agent presents for predicting the consequence of its own behavior. Particularly, it predicts the feature representation of the next state $\hat{\phi}(s_{t+1})$ based on current state s_t and the action a_t taken by the agent. Formally, $\hat{\phi}(s_{t+1}) = f(\phi(s_t), a_t)$. The full picture of the model can be observed in Fig. 4.4.

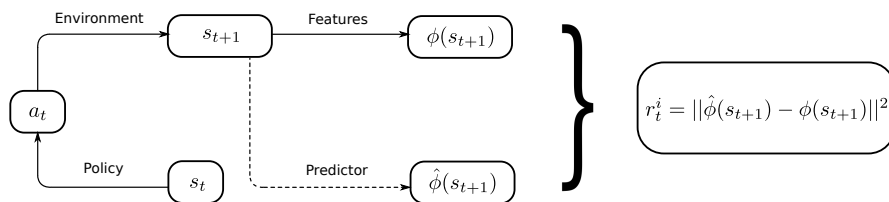


Fig. 4.4.: A predictor based on the forward dynamics of the model is used to estimate the novelty of states from where to derive an intrinsic reward.

The motivation for this method is that given a new state, if similar states have been visited many times in the past, the prediction should be easier and thus the model should present lower prediction error. The exploration bonus is therefore computed as the error between the predicted states,

$$r_t^i = \frac{\eta}{2} \|\hat{\phi}(s_{t+1}) - \phi(s_{t+1})\|_2^2 \quad (4.4)$$

Predicting the forward dynamics presents an unintended feature. In case the environment is highly stochastic, the model receives high intrinsic rewards even though when the state is visited multiple times. This is due to the inability the model presents to predict the results in such cases. Fortunately, CO problems here addressed are deterministic by definition (as argued in Sec. 3.3). Hence, forward dynamics can be applied to estimate the novelty of states.

Inverse Dynamics

Alternatively, the inverse dynamics of the model can also be used to predict the knowledge about the environment. Predicting the forward dynamics is not easy, especially considering that some factors in the environment cannot be controlled by the agent. A good state feature space should exclude such factors because they cannot influence the agent’s behavior and thus the agent has no incentive for learning them. Alternatively, using the inverse dynamics an embedding network is used to predict the action taken by the agent to move from state s_t to s_{t+1} . In that case, the agent will not receive rewards for reaching states that are inherently unpredictable.

As mentioned, currently the RL community has had great result using Prediction-based exploration to solve sparse environments previously intractable e.g. in the famous Sokoban and Montezuma’s revenge games. State-of-the-art works in this field [26, 25, 10] have achieved for the first time positive results in these environments combining long-term predictions with short-term memory models to estimate the novelty of states. Particularly, these works rely on a memory-based short term module to prevent the agent from exploring the same state numerous times inside a episode; and a life-long predictor module that discourage states visited many times across episodes.

In the experimentation, we test a predictor based on the forward dynamics of the problem. This predictor is trained using supervised learning on the current states the model achieves. Building a forward predictor is relatively easy using the features the agent extracts. And as

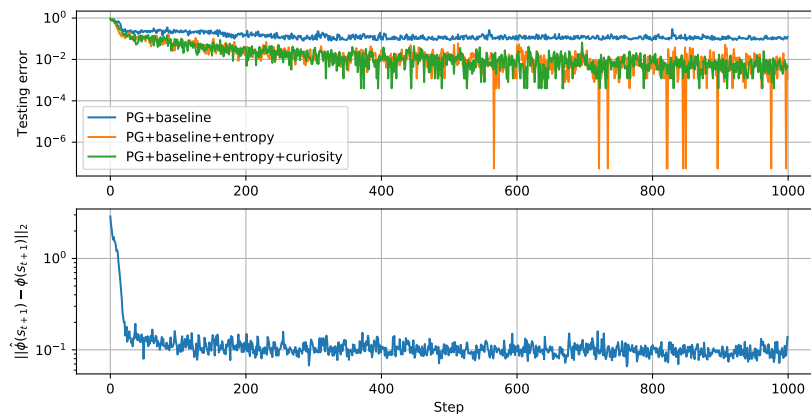


Fig. 4.5.: On top, the testing error experienced on the reference problem without additional exploration techniques, with entropy regularization and curiosity-driven RL based on predicting the forward dynamics of the problem. On the bottom, the prediction error experienced during the learning.

mentioned, in deterministic CO problems, this forward predictor is enough to address the model exploration. The results obtained are depicted in Fig. 4.5. The predictor learns relatively fast a good approximation on the dynamics of the problem. However, we observe that the curiosity-driven mechanism does not add any benefit to the model. In the scenario of CO problems, the effect of Entropy Regularization specially in the long run of the learning is much more considerable than Intrinsic Rewards exploration bonuses. Due to this reason, **only Entropy regularization will be considered as the main exploration method in this work.**

4.2.3 Monotonic improvements with Trust-region optimization

Another technique to improve the convergence in policy optimization methods is Trust-region optimization. In RL, tuning the hyper-parameters correctly is key to obtain the best results on the model. However, during this process some phenomena could arise that prevents it from continuing the learning progression. In particular, in Policy Gradients one of the most relevant occurrences is **policy degradation**. This phenomena can be observed in Fig. 4.6. In this example, the objective function is unstable throughout the learning process, it does not decrease monotonically. This occurs fundamentally due to the presence of large optimization steps done over noise gradient estimations. This noise may be due to different factors ranging from stochastic transitions to exploratory actions. If an update in the policy is large enough, specially in the wrong direction, this can conduct to a degradation in the policy the method does not recover from.

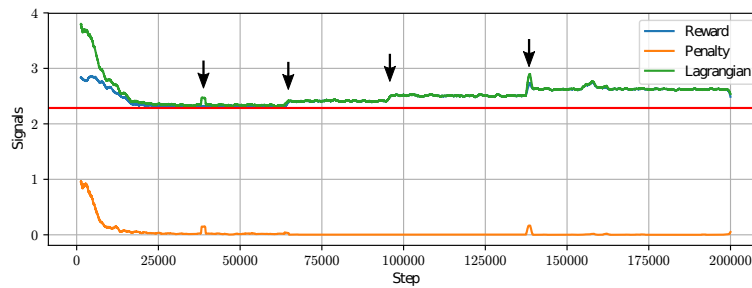


Fig. 4.6.: Example of vanilla-PG presenting performance degradation in the policy learning process. In this picture the red line represents the optimal expected Lagrangian J_L^* computed analytically. As observed, the degradation prevents the model from converging towards a good policy, turning the learning process unstable.

Although the vanilla implementation of Policy Gradients method has been extensively used in NCO, particularly the Monte-carlo Policy Gradients method (also known as the *Reinforce* algorithm) [80], the mentioned limitation is likely to occur on this method. As the objective function L^{PG} is estimated using the empirical average $\hat{\mathbb{E}}_{\pi}$ over a finite batch of samples, from which the optimization algorithm (e.g., stochastic gradient descent (SGD)) performs a step in the convergence. This method often conducts to policy updates in the wrong direction, producing a degradation in the policy. Normally, the optimizer recovers from these wrong updates, but if a change in the learning step is too large, this method might not recover and present a collapse in the performance. This makes difficult to choose a correct learning step-size.

Alternatively, Trust-region optimization can be applied to guarantee that the update in the policy is fitted to the objective function. Trust-region optimization is a well known optimization method that is used in case of having an approximation to the function that is accurate locally but inaccurate far away from the optimization point. This method guarantees that the update stays on the local region, where the approximation is trustworthy. To this end, this technique presents

an constrained optimization sub-problem that has to be solved at each optimization step. Two of the most common adaptations of trust-region optimization to PG are analyzed in the solution, these are Trust Region Policy Optimization and Proximal Policy Optimization.

4.2.3.1 Trust Region Policy Optimization

In **Trust Region Policy Optimization (TRPO)** [98] a surrogate of the PG objective function is maximized subject to a constraint on the size of the policy update. Specifically, this method rewrites the objective function L^{PG} in terms of the change on the policy distribution that the model experience during a learning step. This formulation yields to a resolution process that enables to compute rapid approximations on the trust-region optimization problem.

Particularly, TRPO uses Importance Sampling (IS) (discussed in Annex B.3.2) to obtain this surrogate objective. This technique allows to estimate a policy distribution, relying on samples generated from a different policy. Here, it is used to estimate the objective function under the current policy π_θ using the policy before the update $\pi_{\theta_{old}}$. This allows to define the objective in terms of the difference on the policy update, which is key to this method. The surrogate objective function is obtained as follows,

$$\begin{aligned} \nabla_\theta L^{PG}(\theta) &= \mathbb{E}_{s,a \sim \pi_\theta} \left[\nabla_\theta \log \pi_\theta(A_t|S_t) \cdot A^\pi(S_t, A_t) \right] = \\ &= \mathbb{E}_{s,a \sim \pi_\theta} \left[\frac{\nabla_\theta \pi_\theta(A_t|S_t)}{\pi_\theta(A_t|S_t)} \cdot A^\pi(S_t, A_t) \right] = \\ &= \mathbb{E}_{s,a \sim \pi_{\theta_{old}}} \left[\frac{\nabla_\theta \pi_\theta(A_t|S_t)}{\pi_{\theta_{old}}(A_t|S_t)} \cdot A^\pi(S_t, A_t) \right] \Bigg|_{\theta=\theta_{old}} = \nabla_\theta L^{IS}(\theta) \Big|_{\theta=\theta_{old}} \end{aligned} \quad (4.5)$$

Therefore, the surrogate loss function obtained using importance sampling results in the following equation,

$$L^{IS}(\theta) = \mathbb{E} \left[\frac{\pi_\theta(A_t|S_t)}{\pi_{\theta_{old}}(A_t|S_t)} \cdot A^\pi(S_t, A_t) \right] \Bigg|_{\theta=\theta_{old}} \quad (4.6)$$

The resulting objective function L^{IS} is constrained to guarantee that the update in the policy belongs to the trust region. In particular, TRPO uses the KL divergence to restrict the distance in the policy update. The constraint equation limits the update in the policy distribution to a value δ , that the method configure as an hyperparameter. Formally,

$$\begin{aligned} \max_{\theta} \quad & \mathbb{E}_{s,a \sim \pi_{\theta_{old}}} \left[\frac{\nabla_\theta \pi_\theta(A_t|S_t)}{\pi_{\theta_{old}}(A_t|S_t)} \cdot A^\pi(S_t, A_t) \right] \\ \text{s.t.} \quad & \mathbb{E}_{s,a \sim \pi_{\theta_{old}}} [KL(\pi_{\theta_{old}}, \pi_\theta)] \leq \delta \end{aligned} \quad (4.7)$$

This optimization problem is relaxed using the Lagrange multipliers method. This follows from the fact that a the KL penalized surrogate objective forms a lower bound (i.e., a pessimistic bound) on the performance of the policy π . The penalized objective function states as follows,

$$\max_{\theta} \mathbb{E}_{s,a \sim \pi_{\theta_{\text{old}}}} \left[\frac{\nabla_{\theta} \pi_{\theta}(A_t|S_t)}{\pi_{\theta_{\text{old}}}(A_t|S_t)} \cdot A^{\pi}(S_t, A_t) \right] - \beta \mathbb{E}_{s,a \sim \pi_{\theta_{\text{old}}}} [KL(\pi_{\theta_{\text{old}}}, \pi_{\theta})] \quad (4.8)$$

This non-linear equation is solved using numerical optimization methods. In particular, this problem is efficiently solved approximating it into a well-known expression that uses an affine approximation to the objective and a quadratic approximation to the penalty term. The resulting quadratic program corresponds to the computation of the natural gradient [57, 89]. Nevertheless, its resolution involves the computation of the Fisher Information Matrix, which is intractable to obtain it analytically when large function neural models are used. TRPO avoids this computation approximating the solution using the conjugate gradient. This method succeed in reducing the drops in performance that the policy experience during the learning. However, its implementation is mathematically complex, due to this reason other simpler alternatives are used.

4.2.3.2 Proximal Policy Optimization

On the other hand, **Proximal Policy Optimization (PPO)** [100] has demonstrated to obtain similar if not slightly better results that TRPO with a much simpler implementation. PPO does not use a penalty because it is hard to choose a single value of β that performs well across different problems or even within a single problem, where the characteristics change over the course of learning. Hence, to achieve the goal of a first-order algorithm that emulates the monotonic improvement of TRPO, experiments have shown that it is not enough to simply choose a fixed penalty coefficient β and optimize the penalized objective equation.

In their work [100], the authors introduce two different alternatives to TPPO:

- **PPO-Penalty**: that approximately solves a KL-constrained update like TRPO, but penalizes the KL-divergence in the objective function instead of making it a hard constraint, and automatically adjusts the penalty coefficient over the training in order to be appropriately scaled.
- **PPO-Clip**: that instead of dealing with the constraint, it relies on specialized clipping in the objective function to remove incentives for the new policy to get far from the old policy.

This Thesis resort in the last option PPO-Clip, to limit the objective from an excessively large policy update. To this end, this technique establishes a maximum in the distance that the ratio between the target policy and the previous policy can have. The main objective PPO-Clip is defined in the following expression:

$$\begin{aligned} L^{\text{CLIP}}(\theta) &= \mathbb{E}_{s,a \sim \pi_{\theta}} \left[\frac{\nabla_{\theta} \pi_{\theta}(A_t|S_t)}{\pi_{\theta_{\text{old}}}(A_t|S_t)} \cdot A^{\pi}(S_t, A_t) \right] = \mathbb{E}_{s,a \sim \pi_{\theta}} [\psi(\theta) \cdot A^{\pi}(S_t, A_t)] \approx \\ &\approx \mathbb{E}_{s,a \sim \pi_{\theta}} [\text{CLIP}(\psi(\theta)(1 - \epsilon, 1 + \epsilon)) \cdot A^{\pi}(S_t, A_t)] \end{aligned} \quad (4.9)$$

where ψ denote the probability ratio and ϵ the clipping range hyperparameter. PPO modifies the surrogate objective by clipping the probability ratio, which removes the incentive for moving ψ

outside of the interval $[1 - \epsilon, 1 + \epsilon]$. Finally, it takes the minimum of the clipped and unclipped objective, so the final objective is a lower bound (i.e., a pessimistic bound) on the objective.

PPO in its clipped objective variant is extremely easy to apply over the PG algorithm. In addition, it outperforms TRPO in some environments [100]. Due to this reason, this method is gaining popularity in the community and it is the learning algorithm, among others, that conform the final solution on our proposal.

4.2.3.3 Implementing Trust-region optimization using OpenAI framework

As mentioned, Trust-region optimization can be applied to prevent that the update is fitted to the objective function, and thus avoid large updates that collapse the performance. Trust-region optimization is a well known technique that is used in case of having an approximation to the function that it is only accurate locally. As this method guarantees that the updates stay on the local region, where the approximation is reliable.

There are two tightly linked algorithms that employ trust-region optimization: Trust-Region Policy Optimization (TRPO) and Proximal Policy Optimization (PPO). In this section, a performance comparison between the mentioned RL algorithms is done to test their ability to prevent policy degradation. To facilitate this process a RL framework is used. RL frameworks provide tested high-quality implementations of state-of-the-art RL algorithms. Specifically, **OpenAI baselines RL framework** [36] is used for this purpose, as it includes a well documented implementations of the referenced methods. In particular, OpenAI is a TensorFlow-based framework that provides a complete package for training models compatible with the OpenAI Gym interface [24].

Using the OpenAI framework, implementations of TRPO (Sec. 4.2.3.1), PPO (Sec. 4.2.3.2), Advantage Actor-Critic (A2C) (Annex B.5) and DQN (Annex B.3.2) algorithms are tested in this simple toy CO problem. The comparison in the learning process between these methods is shown in Fig. 4.9 and the configuration parameters in Tab. 4.1. The results show that **PPO behaves better in this CO problem**. Overall, we observe that policy-based methods (PPO, TRPO, A2C) perform better in this kind of problems, whereas it is challenging for value-based methods (DQN) to approximate a value function in this context. Hence, we would particularize the solution for policy-based algorithms.

Analyzing the resulting learning process (see Fig. 4.7), one observes that using PPO the degradation in the policy becomes to almost negligible. The performance losses derived from aggressive updates in the policy disappear. We conclude therefore that PPO significantly reduces the performance degradation experienced in the previously used vanilla PG algorithm. This enables a monotonic optimization process that leads to a optimization and ultimately to a improvement in the behaviour achieved. All in all, this method reduces significantly the testing error previously experienced (in this test approximately in a couple of orders of magnitude). However, a residual testing error is still present in our search for the optimal policy.

Details on the OpenAI model implementation and learning process

OpenAI baselines is a reference framework in the RL community that contributes with quality implementations of RL algorithms. However, this software emphasizes being compact rather than comprehensive and easy to integrate. Due to this reason, a branch of this framework arose from the academia with the purpose of unifying the code and facilitate its utilization, this is Stable baselines [51].

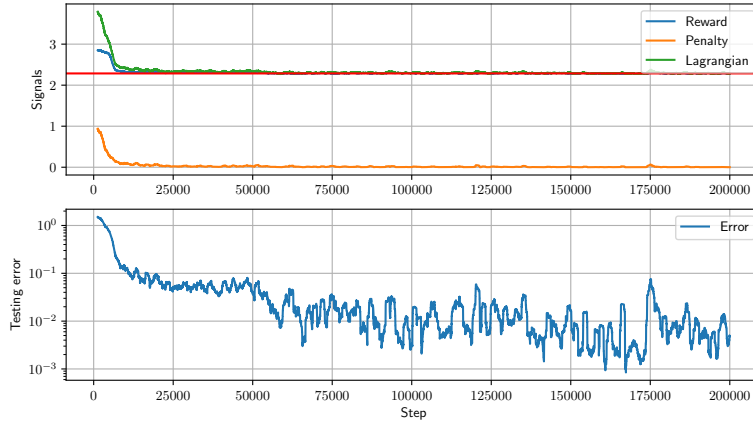


Fig. 4.7.: Learning process on the PPO implementation. In this picture the *red* line represents the optimal Lagrangian J_L^* computed analytically. As observed, the performance of the model increases monotonically, the optimization process obtains better policies and the testing error improves.

Creating a custom environment in Stable baselines requires to satisfy some technical specifications, but once the problem is integrated in the framework, models can be trained seamlessly with the different learning algorithms the software provides. In particular, this framework works with environments compatible with the OpenAI Gym interface¹. In addition to lay out this specific interface, in case of requiring an specific neural agent not included in the framework, it shall be designed. As it is the case of the model-specific agent we pursue. However, this framework imposes some restrictions in this sense, e.g., the definition of the loss function shall be unique for the model. To address this issue, complex agents as actor-critic architectures are required to be built over a multi-headed neural network (depicted in Fig. 4.8).

This multi-headed network partially shares the weights of the actor and critic networks. That is to say, both agents share the state representation, although they rely on separate heads to produce the policy and the value functions respectively. This is justified as first layers correspond to basic features extraction, features that are likely to be the same for the policy and value approximators. Leaving to the last layers the ability to produce the specific results that each head requires. Here, the implementation of the loss function is done weighing the policy gradients loss in the actor together with the loss on the value function estimator or critic. This requires of an additional hyperparameter ν that estimates the weight of each model on the final loss function. As entropy regularization is used to improve the exploration in this test then an entropy loss term is also included. This results in the following formula,

$$loss = PG_{loss} + \beta \cdot \mathbb{H}_{loss} + \nu \cdot VF_{loss} \quad (4.10)$$

Although this technique reduces the number of training parameters, the resulting multi-criteria objective results in additional plateaus, and in the end, a harder function to optimize.

Lastly, the evolution of relevant learning indicators on the PPO algorithm are included in Fig. 4.10. Understanding these internal indicators is key to control the evolution of the learning process and perform a correct tuning of the hyperparameters. Unlike in ML, where some grid search methods are commonly used for this purpose, in RL training a model generally last a much longer

¹An **OpenAI Gym** blueprint can be found in: <https://github.com/rubensolozabal/OpenAI-Gym-demo-environment>

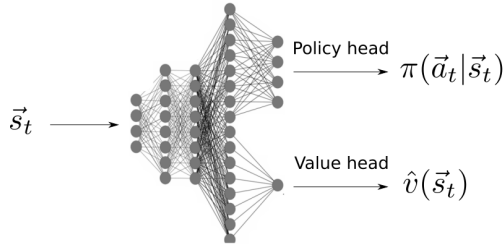


Fig. 4.8: Actor-critic agent embedded in a multi-headed network. In the model, both the static definition of the CO problem \mathbf{x} and the dynamic state of the environment d_t are concatenated to create the state vector \vec{s}_t from where the policy $\pi(\vec{a}_t | \vec{s}_t)$ and value estimator $\hat{v}(\vec{s}_t)$ heads are computed sharing the neural network.

period of time. Due to this reason, search methods are not a viable option and the selection of the hyperparameters relies on the expertise of the engineers.

Analyzing these learning indicators, some conclusions can be drawn to identify if the learning process is well under way. First, observing the policy gradients loss (*policy_gradient_loss*) and value function loss (*value_function_loss*) one can evaluate the convergence on both approximators, the actor and the critic. Also, as this model uses entropy regularization, one can control the amount of entropy added throughout the leaning. This is done via the entropy coefficient β . The amount of entropy added to the model can be observed in the entropy loss function (*entropy_loss*). Another signal that it is important to monitor is the KL divergence of the policy (*approximate_kullback_leibler*). Ensuring that the model does not present spikes in this indicator is key to prevent policy degradation. In PPO this behaviour is controlled using the clipping-range parameter ϵ . The strength in which the clipping is actuating over the loss function is reflected in the clipping factor function (*clip_factor*). E.g., large values of the clipping factor indicates that the model is consistently reducing the size of the updates, in those cases it is desirable to directly decrease the learning step. Finally, monitoring the norm of the gradients (*grad_norm*) is also important as it helps to identify the progression in the convergence of the model. In this case, the norm of the gradients is also clipped to a max value. Gradient clipping is a technique that prevents large gradients and enhances the convergence [128].

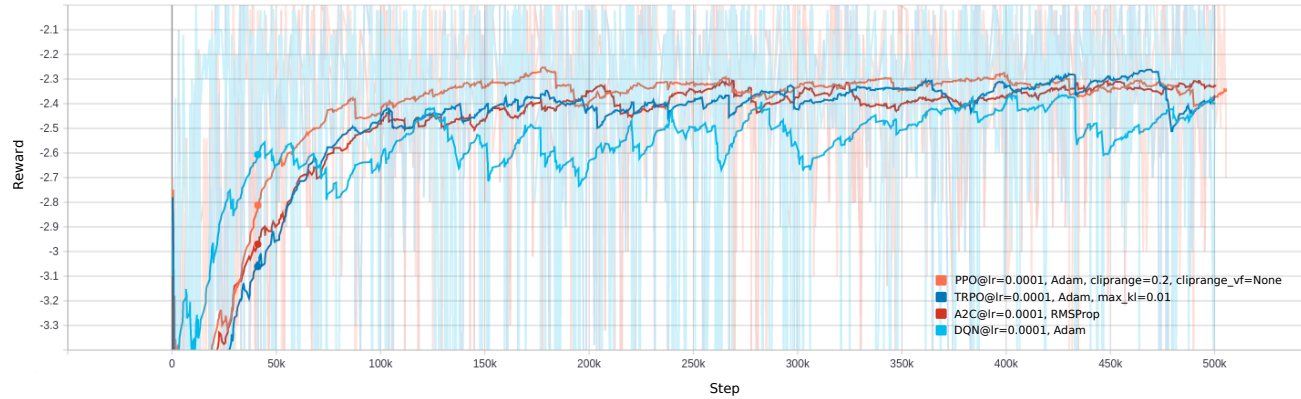


Fig. 4.9.: Performance comparison between PPO, TRPO, A2C and DQN in a demo combinatorial problem built on OpenAI Stable Baselines framework.

Tab. 4.1.: Hyperparameter configuration.

	PPO	TRPO	A2C	DQN	Description
gamma	1	1	1	1	Discount factor.
batch_size	128	128	128	128	The number of steps to run for each batch per update.
ent_coef	0.01	0	0.01	-	Entropy coefficient for the loss calculation.
learning_rate	1E-04	-	1E-04	1E-04	The learning rate.
vf_coef	0.5	0	0.5	-	Value function coefficient for the loss calculation.
max_grad_norm	0.5	-	0.5	-	The maximum value for the gradient clipping.
lamda	1	1	1	-	Factor for trade-off of bias vs variance for Generalized Advantage Estimator.
cliprange	0.2	-	-	-	Clipping parameter for the policy estimator.
cliprange_vf	0	-	-	-	Clipping parameter for the value function.
max_kl	-	0.01	-	-	The maximum value for the gradient clipping.
conjugate_iters	-	10	-	-	The number of iterations for the conjugate gradient calculation.
optimizer	Adam	Adam	RMSprop	Adam	Optimization algorithm.
beta1_optimizer	0.9	0.9	-	0.9	Exponential decay rate for the first moment estimates.
beta2_optimizer	0.99	0.99	0.99	0.99	Exponential decay rate for the second moment estimates.
momentum	-	-	0	-	RMSProp momentum parameter.
prioritize replay	-	-	-	False	If True prioritized replay buffer will be used.
exploration prob.	-	-	-	1 ->0.02	Probability of random action exploration.

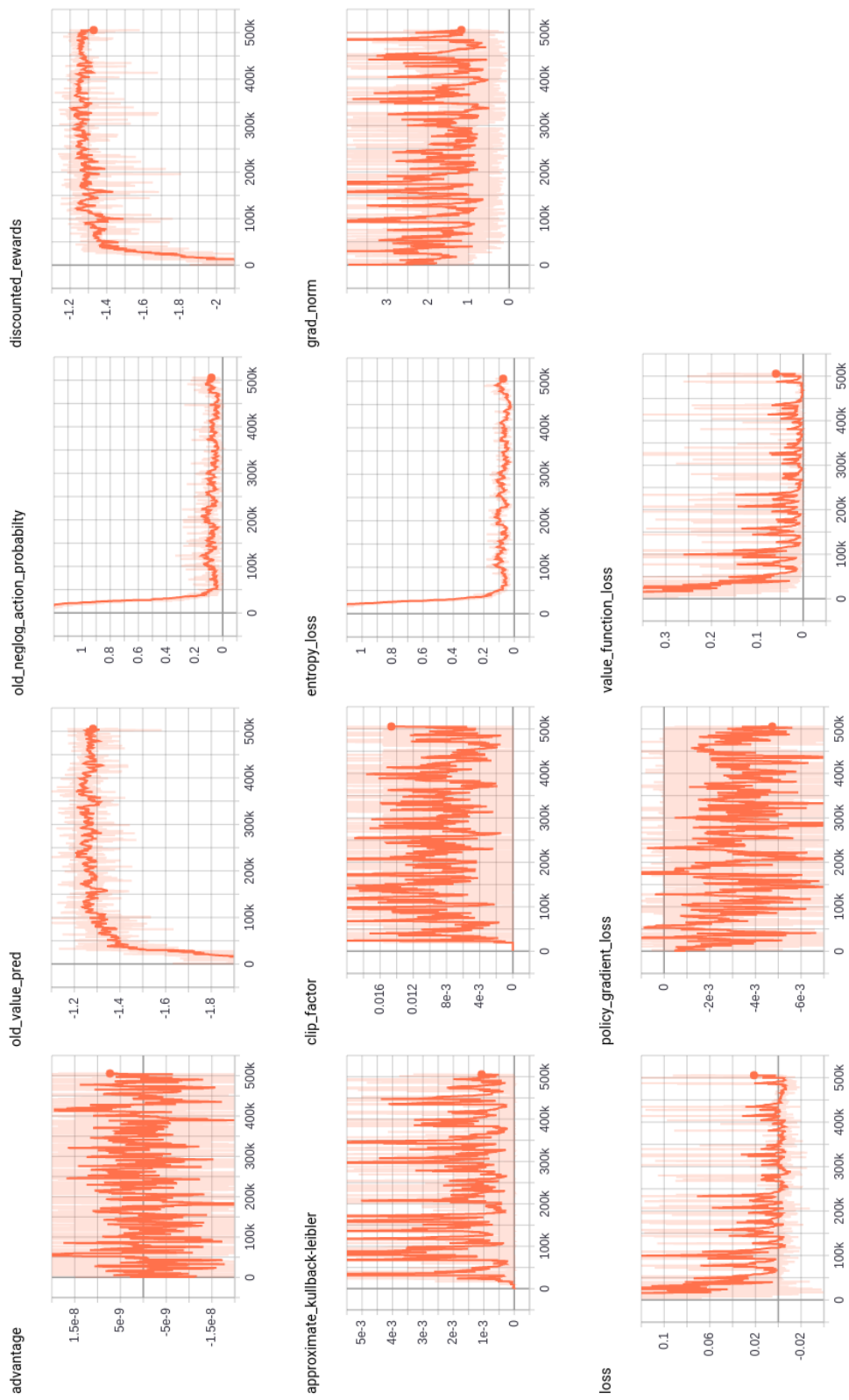


Fig. 4.10.: Learning key indicators extracted from the OpenAI framework on the PPO implementation. Analyzing the evolution of the internal parameters of the model is of utmost importance to control the learning process and ensure a good result.

4.2.4 Improving the model with a self-competing strategy

Despite the positive results obtained in the OpenAI implementation, RL frameworks present some limitations. **RL frameworks in general put compatibility above performance.** In this direction, these frameworks usually build a common interface to communicate with the different CPU-oriented environments the user integrates. However, this CPU-oriented strategy in the problem description is not the most efficient approach. Alternatively, a fully-vectorized implementation can be built to benefit from tensor-oriented hardware and thus, accelerate the learning process. In addition, this approach will allow us to raise a novel learning strategy, a self-competing scheme in which the model competes against itself to improve its policy by promoting the best results it obtains.

Benefits of a fully-vectorized implementation

In this occasion, a fully-vectorized implementation of the problem is built, and thus, high-level RL frameworks are not required. Commonly, due to the computational effort that backpropagation requires, the agent is coded using an automatic differentiation library compatible with GPU or TPU hardware that accelerates these tensor operations, whereas the environment generally runs on the CPU. As RL consists in a constant iteration between an agent and the environment, the execution cycle repeatedly involves a **context switch between the CPU and GPU/TPU units** (see Fig. 4.11). This context switch is usually synchronous, which means that while one module is operating, the other needs to wait for the results to continue. This architecture is commonly used in RL frameworks due to the ease of integration, as only a suitable interface in the environment is required to integrate them into the model. However, and despite all major RL frameworks try to alleviate this phenomena including paralization via CPU threads, this approach severely slows down the learning process.

As an alternative, here, the complete model (agent and environment) is built in a fully-vectorized implementation that runs on tensor accelerated hardware. Building CO problems as tensor-oriented operations is not straightforward and even in some frameworks this cannot be done. E.g., TensorFlow before the Eager² execution was integrated, the graph of tensors was fixed in execution and operations could not be modified in operational time. This made very difficult to include conditional operations on the model and thus was a non-viable framework to integrate complex environments. It was with PyTorch [86], and later with TensorFlow 2.0, that dynamic computational graphs were included, turning full-vectorization into a viable option.

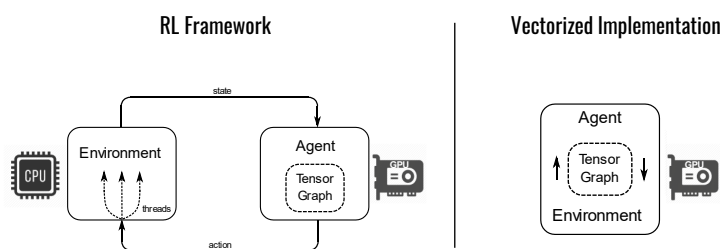


Fig. 4.11.: Hardware implementation of the RL model. On the left, the traditional approach that combines the usage of CPU and GPU to compute the model. On the right, a fully-vectorized implementation that benefits from tensor-accelerated hardware in both, the agent and the environment.

²TensorFlow Eager execution implements imperative programming, this enables to evaluate operations immediately without building operational graphs.

Self-competing baseline estimator model

One can benefit from this fully-vectorized implementation to integrate variance reduction techniques without requiring an additional estimator. As mentioned, due to the performance observed in the experimentation (Sec. 4.2.3), PG-based methods are the selected option to address the problems ahead. Particularly, the PPO algorithm is the one that presents the best trade-off between performance and implementation complexity. And thus, it is the selected learning algorithm. In PG-based methods, the variance of the gradient computation is sometimes excessive. This is because complete trajectories are used to compute the return (see Annex B.4.1). This variance in the results produce a slow and unreliable convergence on the optimization process. Due to this reason, there has been much effort devoted to develop variance reduction techniques that mitigate this effect.

A common method to reduce the variance is to introduce a baseline function $b \rightarrow \mathbb{R}$ in the reward term. This reduces the variation in the returns while still maintains an unbiased estimator. The proof on why the inclusion of the baseline b term does not introduce bias in the gradient and reduces the variance can be found in Annex B.4.2.

In short, the baseline b estimates the reward the model achieves for a problem instance s , so that the current result obtained for the instance $L^\pi(a|s)$ can be compared to the performance of the policy π . The baseline is usually build as an estimator that can be as simple as a moving average $b(s) = M$ with decay β , where M equals L^π in the first iteration, and updates as $M \leftarrow \gamma M + (1 - \gamma)L^\pi$ in the following ones. A popular alternative is the use of a learnt value function or «critic» $\hat{v}(s, \theta_\phi)$, where the parameters θ_ϕ are learnt from observations [47]. The baseline performs in the following way, the advantage function $L^\pi(a|s) - b(s)$ is positive if the sampled solution is better that the baseline, causing these actions to be reinforced, and vice-versa. Here, **a new baseline is proposed based on estimations over the current stochastic policy. This method allows to not rely on an additional estimator for computing the baseline.**

In this approach, the learning batch B is increased introducing N times every instance of the problem. Since during the learning process the policy is stochastic, for every instance it obtains N different solutions. This creates a reward distribution Q_j^N that is used to estimate the current performance of the model on the instance x_j . In particular, the baseline estimator $b(s)$ is selected as the quantile (e.g.: $a = 0.1$) of the obtained distribution. The baseline is therefore calculated as

$$b(s_j) = \{q_j : Pr(Q_j^N \leq q_j) = a\} \quad (4.11)$$

This model presents several benefits. First, as it is a fully-vectorized implementation, the size of the learning batch can be increased without a affecting the learning time. This is due to any

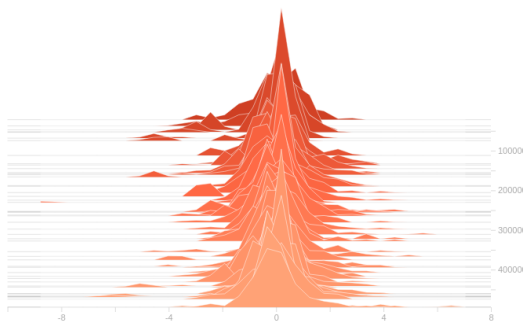


Fig. 4.12: Temporal histogram of the normalized advantage estimator. The advantage estimator is continuously controlled during the learning process in mean and std in order to provide an stable learning reference. That means that half of the samples would provide a positive impact on the policy in a normalized amplitude, whereas the other half of the samples would have a negative repercussion.

part of the execution cycle is computed sequentially. Increasing the size of the batch reduces the variance and improves the convergence, although the amount of memory required to execute the graph also increases. In addition, this model do not require to build a secondary network to estimate the baseline, which results in a simpler implantation.

The results using this model are depicted in Fig. 4.13. As mentioned, for an instance of the problem N solutions are obtained in parallel, and from all of them the one with the best result is selected. The figure represents the testing error of the solution with less error, but also the mean and maximum error achieved in the distribution. Furthermore, an optimality study is made. The percentage of the N solutions for each instance of the batch that achieve the optima is depicted. Although this method improves the results obtained, as can be seen, at the end of the learning process, there are still instances in which none of the N solutions achieve the optima.

A conclusion on why these models do not achieve the optima for all cases even in small combinatorial problems is the following one. For RL models it is asymptotically difficult to update the weights in the right direction as the policy closes to the optimal behaviour. The closer to the optima the model is, the weaker the impulses to improve the policy are. All the techniques seen so far help to improve the model, which is beneficial specially when the problem scales up, as it will be discussed in the experimentation. But an optimal solution for all instances is asymptotically difficult to be achieved.

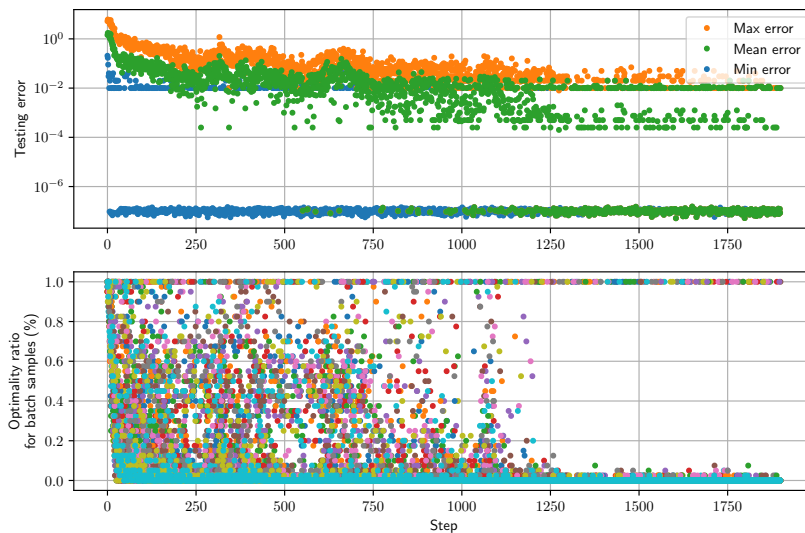


Fig. 4.13.: Testing error and optimality ratio for the Self-competing baseline based on multiple estimations for each instance of the problem.

4.2.5 Search strategies

As argued in (Bello et al., 2016)[15], the bare results obtained from the agent can be improved applying search methods at inference. As evaluating the solutions in these problems is inexpensive, the agent can simulate a search procedure at inference time by considering multiple candidate solutions and selecting the best one. Some common techniques are: a greedy inference over multiple trained models, the sampling technique and the beam search. In the first one, multiple models are learnt, and at inference time the greedy output from every model is evaluated to select the best one. Another alternative is sampling, which consists in improving the exploration at inference using a temperature hyperparameter T to control the sparsity of the output distribution.

In this case, multiple samples are taken and the best of them is selected as the output. This method is used to prevent the model from being overconfident, allowing to evaluate at inference proximal policies. Lastly, beam search is an strategy that has obtained good results in natural language processing. It explores the solutions by expanding the most promising paths in a limited set.

Although search strategies improve the performance of the model, they add a significant cumulative time to the inference process [15]. This makes these solutions incompatible with the real-time decision-making system pursued in this work. Due to this reason, **the results given in the experimentation are the bare solutions the model achieves and none search strategy is used to improve the results at inference.**

4.3 Proposed framework for addressing NCO

During the building process carried out on in this Chapter, the different components that form the model have been settled. Here, these components are all put together to define the learning framework to be used in the experimentation.

Starting with the learning algorithm, the analysis performed in Sec. 4.2.3 has revealed that PPO represents a good strategy due to the positive trade-off that presents between the results obtained and the complexity to be implemented. This learning algorithm is going to benefit from the *self-competing strategy* used in the fully-vectorized implementation of the model we envision (Sec. 4.2.4). This self-competing strategy accelerates the learning process while reduces the variance. Regarding the neural agent, the Markovian models discussed in Sec. 3.4.2 are used. These alternatives utilize different encoding strategies to embed the instance of the problem (recurrence or fully-attention layers). A comparison between both options is going to be carried out on the experimentation. Finally, the proposed framework avoids the use of post-processing methods to improve the solutions at inference time, in favour of enhancing learning strategies that focus on improving the performance of the model itself. The following strategies are going to be considered: entropy regularization (Sec. 4.2.3) and the sampling strategy intrinsic in the self-competing strategy (Sec. 4.2.5). On this framework, adaptive momentum optimizers are used to perform a PG-based learning. The overall picture can be observed in Fig. 4.14.

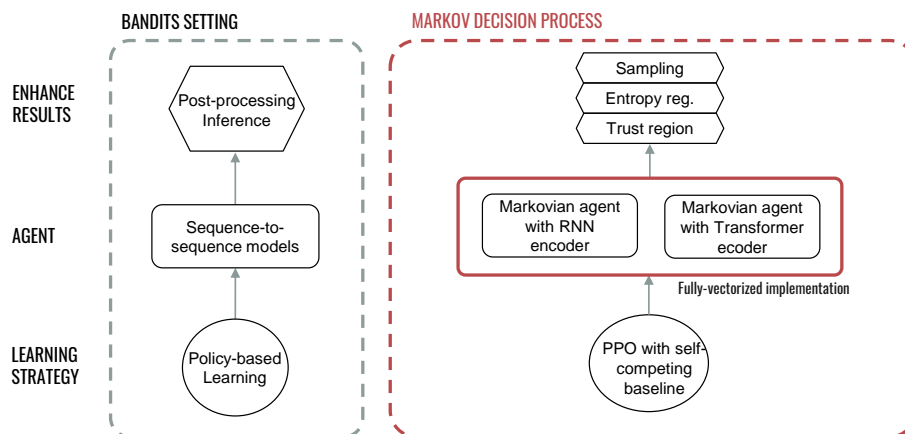


Fig. 4.14.: Testing error and optimality ratio for the Self-competing baseline based on multiple estimations for each instance of the problem.

4.4 Experimentation on the Job-shop Scheduling Problem

In order to validate the proposed framework, we optimize a classical and well-known constrained combinatorial problem, the **Job-shop Scheduling Problem (JSP)** [41, 27]. For this problem, there exist a huge number of variants in the literature. For the sake of evaluating the potential of the proposed model, we select a variant that presents both types of constraints we argue in this Thesis: constraints that can be embedded into the model, and constraints that need to be relaxed and incorporated into the objective function.

4.4.1 Job-shop Scheduling Problem with limited idle time

In the Job Shop Problem (JSP) there exist a number of n jobs $J = \{J_0, J_1 \dots J_{n-1}\}$ and a set m machines $M = \{M_0, M_1 \dots M_{m-1}\}$. Within every job J_i there is a number of operations O_i that need to be processed in an specific order $O_i = \{O_{i,0}, O_{i,1} \dots O_{i,m-1}\}$. For each operation $O_{i,j}$, the machine $M_{i,j}$ and the duration time $D_{i,j}$ associated are defined. The aim in this problem consist of assigning the jobs to the machines such that the total length of the operations period is minimized. This objective is also known as the *makespan*. The classical JSP presents two types of constraints:

- Precedence constraints: specify that for every two consecutive operations in a job, the first one must be completed before the second one can be scheduled.
- No overlap constraints: these constraints arise from the fact that a machine can only work in one operation at a time.

These constraints can be managed via a masking scheme, so we implement them as hard-constraints in our model. In order to include non-maskeable restrictions, the JSP variant with limited idle time was considered. Under that constraint, for any machine, the period between finishing an operation and starting the next operation (idle time) cannot exceed a certain threshold T_{th} . This constraint arises naturally in real context, as the aim is usually to maximize the productivity of the machinery.

In the JSP with limited idle time, the objective function (*makespan*) can be penalized by the sum of all intervals in which the idle time between operations exceed the threshold T_{th} . Hence, the function to minimize is defined as

$$L = \max \mathcal{M}_i + \lambda \sum_{i,j} ((t_{O_{i,j}}^{start} - t_{O_{i,j-1}}^{end}) - T_{th})^+ \quad (4.12)$$

where \mathcal{M}_i denotes the time until the job J_i is finished, and $t_{O_{i,j}}^{start}, t_{O_{i,j}}^{end}$ the start and ending time scheduled for the operations.

4.4.2 Particularized models

Two different models are proposed for addressing the JSP problem: (a) a recurrent encoder-based model and (b) a fully-attentional model. These are depicted in Fig. 4.15-4.16 respectively. The first model (a) uses recursion to encode the information of the sequence of operations that compose each job, yet the solution is iteratively computed. The second one (b) is a fully-attentional model based on a Transformer encoder. In both cases, at each time-step t , the models compute a binary

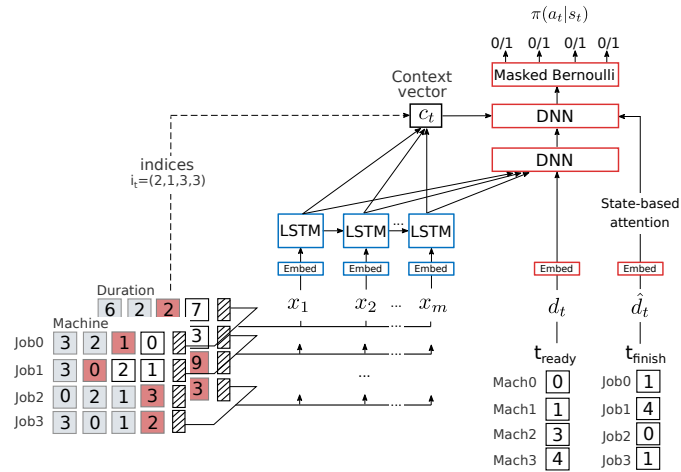


Fig. 4.15.: Neural model particularized for the JSP problem. The model is formed by a single LSTM encoder that operates over the sequence of operations for each job. The resulting vector that describes the combinatorial problem is combined with the state of the environment to decide the operations to be scheduled at each time-step t .

action deciding whether the next operation for each job is scheduled. To this end, the model stores an index vector i_t pointing at the operations that are required to be scheduled next. Notice that the operations for a job must be assigned in a specific order, that is an operation cannot be scheduled until the previous one has finished. This procedure is repeated until all operations are assigned.

As introduced, an instance of the JSP problem is defined by the machine assignment M_{ij} and the time duration D_{ij} matrices. For each operation O_{ij} , these values are concatenated to create the input static feature vector, denoted as x_{ij} . This vector is embedded and further encoded. In the case of using a recurrent encoder, the codification process is configured backwards for this problem and it produces for each operation a representation of the remaining operations until the job is completed. We refer to this vector as $e_{ij} = enc(x_{ij}, \dots, x_{im}) \forall i$. Whereas, the Transformer encoder does not use recursion to embed the sequence of operations. It relies on a positional encoder and a self-attention mechanism to provide direct access on the input to the decoder.

The state of the environment is defined by the state of the machines and the operations currently being process at the decision time. We represent the state using two vectors: the first one, indicates the number of time units until the machines are released; and the second one, the time left for the previous operation to finish. Those vectors constitute the dynamic part of the input d_t , and are recomputed at each time-step t .

Both parts, the static and the dynamic state, are concatenated to create the input $s_t = (\mathbf{x}, d_t)$ from where the DNN computes the output probability distribution (depicted in red in Fig. 4.15 - 4.16). In this example, the output corresponds to a Bernoulli distribution, that indicates for each job whether the current operation (pointed by i_t) should be scheduled. Nevertheless, not every action can be selected at any time. Actions that lead to an infeasibility in the *precedence* or *no overlap constraint* are masked. This is achieved forcing to zero the probability of scheduling the operation. In order to build the mask, the required information indicating whether the previous operation has finished or a machine is free to use, this can easily be gathered from the state vector d_t .

Finally, the model presents a double attention mechanism: one on the operations to be scheduled, and another over the state representation. The first one, corresponds to the context vector c_t . In

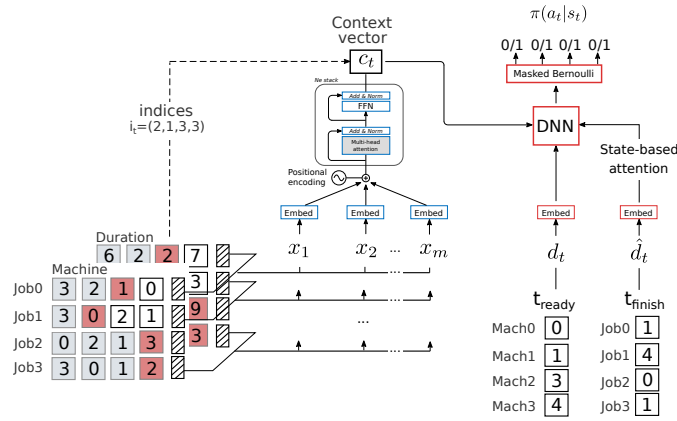


Fig. 4.16.: Fully-attentional model particularized for the JSP problem. The model uses a Transformer encoder to directly attend over the sequence of operations for each job. The self-attention mechanism is combined with the state of the environment to decide the operations to be scheduled at each time-step t .

the case of the recurrent model (a), this vector gathers from e_{ij} the indices pointed by the vector i_t . It acts, as a glimpse on the operations yet to be scheduled in each job. In the fully-attentional model (b), the context vector is an attention vector that introduces the current operation i_t into the attention mechanism. The second attention vector corresponds to glimpse on the time for the previous operation to finish \hat{d}_t . This vector is introduced deeper into the model to enhance the feature extraction on which the solution is computed.

4.4.3 Learning algorithm: PPO with self-competing baseline

In regard with the learning method used to implement the reward constrained policy optimization, it is a PPO with self-competing baseline. Particularly, a Monte-Carlo Policy Gradients with PPO clipping. This includes the self-competing baseline described in Section 4.2.4. As argued in this work, a single agent is employed to learn a policy π_θ that acts as an heuristic for solving constrained combinatorial problems. In the learning process a set of B instances are sampled from the problem distribution \mathcal{S} . The set is computed N times to estimate the objective distribution the policy presents for each instance. This procedure allows to generate a baselines estimator relying on the different results that the stochastic policy produces during the learning process. A critic network is not required to produce a low-variance estimation of the reward. We call to this method self-competing baseline, as the model reinforces the best solutions the stochastic policy gets. The learning algorithm is described in Algorithm 1.

4.4.4 Results on the Job Shop Problem

In this section, we present the experimental study on the classical Job Shop Problem ($\lambda = 0$) and also on the *limited idle time* variant ($\lambda > 0$). The results obtained on our framework are compared with a Generic Algorithm (GA) [29] and the solver CP-SAT from OR-Tools [88]. In addition, in the case of the classic JSP results are also compared with some well-know heuristics: the *Shortest Processing Time* (SPT), *Longest Processing Time* (LPT), *First-come-first-served* (FCFS) and *Least Work Remaining* (LWR) [71].

In the experimentation, two different encoding mechanisms are used: LSTM neural networks in the recurrent encoder and the Transformer encoder of [116] in the fully-attentional model. The learning algorithm has previously been presented (Sec. 4.4.3) and the objective function is

Algorithm 1 PPO with self-competing baseline

```
1: Initialize the actor network with random weights  $\theta$ 
2: for episode = 1,2,... do
3:   Sample problem instance  $\mathbf{x}^j \in \mathcal{X}$  for the batch  $j \in \{1, \dots, B\}$ 
4:   for j = 1,...,B do
5:     for n = 1,...,N do
6:       Initialize step counter:  $t \leftarrow 0$ 
7:       repeat
8:         Sample action  $a_t^{jn}$  from the output distribution  $\pi_\theta(\cdot | s_t^{jn})$ 
9:         Observe the state  $d_t^{jn}$ 
10:        Create the input for the next step  $s_{t+1}^{jn} = \{\mathbf{x}^j, d_t^{jn}\}$ 
11:         $t \leftarrow t + 1$ 
12:       until termination condition is satisfied
13:       Compute the objective function once the solution is obtained  $L(y^{jn} | s^j)$ 
14:     end for
15:     Create the objective distribution  $Q^j$  with the  $N$  samples obtained for the problem instance  $s^j$ 
16:     Compute the baseline:  $b(s^j) = \{q^j : Pr(Q^j \leq q^j) = a\}$ 
17:   end for
18:   Compute the gradient:  $g_\theta = 1/(B \cdot N) \cdot \sum_{j=1}^B \sum_{n=1}^N clip(L(y^{jn}) - b(s^j), \epsilon) \cdot \nabla_\theta \log \pi_\theta(y^{jn} | s_j)$ 
19:   Update the weights:  $\theta \leftarrow \text{Adam}(\theta, g_\theta)$ 
20: end for
```

optimized using the Adam optimizer. Further details the implementation hyperparameters can be found in Appendix C.

In regard with the decoding mechanisms, a greedy and a sampling technique are tested. In the greedy approach, the solution is directly obtained from the model, whereas in the sampling method, multiple solutions are computed from the stochastic policy, and the best one is selected. This latter method comes natural within the self-competing strategy proposed and it does without adding overhead to the model. These instances are referenced as RL_S for the recurrent model and RL_T for the attentional model, followed by the number of solutions taken in the experiment.

As mentioned, the results of the model are compared with those obtained by OR-Tools. For small size instances, the solver is able to compute the optimal solution. However, for larger instances or when the number of restrictions is higher, as it is the case of the *limited idle time* variant, computing the optimal solution becomes intractable. In those cases, we limited the execution time up to one hour, and the solutions obtained are only considered as near-optimal approximations. Also, a GA is tested to provide a vision on how metaheuristics perform in the problem.

The results are summarized in Tab. 4.2. It introduces the average objective, the standard deviation and the mean computation time obtained by the different methods for the classic JSP ($\lambda = 0$) and the JSP with *limited idle time* ($\lambda = 1$). Performance measures were averaged on a set of 50 instances for each problem size³.

We observe that in the classic JSP, the recurrent model is competitive in terms on the quality of the solution when compared to heuristics and the GA, specially for small and medium problems instances (JSP10x10 and JSP15x15). Moreover, the variance obtained in the model is considerable low during the tests. We conclude, therefore, that the model is robust in the sense that the results are consistent in performance. It can also be observed that the sampling technique RL_S(40) provides a reasonable tradeoff between the computational cost and the improvement in the results. It is therefore, sampling size we used hereof. On the other hand, the attentional model is not a viable alternative on this problem. Due to the large state representation the problem requires,

³The size of the JSP is defined by the number of jobs n and the number of machines m . We therefore denote the problems as JSP $n \times m$.

Tab. 4.2.: Average objective, standard deviation and mean computing time for instances of the *classic* JSP ($\lambda = 0$) and JSP with *limited idle time* ($\lambda = 1$). The size of the instance is denoted by the number of jobs n and the number of machines m : JSP $n \times m$.

Method	JSP10x10			JSP15x15			JSP20x20			JSP25x25			
	($\lambda = 0$)	mean	std	time	mean	std	time	mean	std	time	mean	std	time
SPT		99.9	9.1	0.005s	153.2	10.5	0.012s	198.3	9.3	0.022s	252.9	13.2	0.045s
LPT		107.8	9.7	0.005s	163.9	10.9	0.012s	218.8	13.7	0.023s	278.2	17.2	0.050s
FCFS		107.1	10.0	0.005s	163.2	13.7	0.012s	219.3	13.1	0.023s	276.9	14.7	0.051s
LWR		113.9	14.1	0.037s	174.8	12.6	0.123s	227.3	12.7	0.279s	287.2	18.0	0.598s
GA_P(300)		96.4	5.2	55.80s	169.4	6.7	165.8s	254.3	7.2	303.9s	338.2	7.6	586.0s
RL_S(1)		101.3	8.5	0.83s	161.9	11.9	2.15s	216.2	14.4	3.56s	277.2	17.3	5.16s
RL_S(40)		91.9	5.8	1.04s	143.6	6.4	2.31s	196.9	7.7	4.38s	249.3	7.7	6.38s
RL_S(100)		90.7	5.4	1.17s	142.1	6.6	2.65s	193.6	8.0	4.52s	244.5	8.1	7.04s
RL_T(40)		92.2	9.3	1.61s	-	-	-	-	-	-	-	-	-
OR-Tools		81.5	4.6	0.082s	118.8	4.4	61.22s	156.2	4.5	1h(*)	195.4	4.9	1h(*)
($\lambda = 1$)	mean	std	time	mean	std	time	mean	std	time	mean	std	time	
GA_P(300)	343.7	45.5	91.6s	1117.0	39.0	257.3s	2476.0	62.6	578.4s	4453.2	111.7	1079s	
RL_S(40)	348.1	36.6	1.36s	860.2	65.4	3.12s	1573.0	112.7	5.18s	2745.0	173.1	7.75s	
OR-Tools	221.4	18.4	1h(*)	593.5	19.9	1h(*)	1221.0	50.9	1h(*)	2075.0	101.2	1h(*)	

(*) The result is not optimal, the execution has been forced to end after the indicated time.

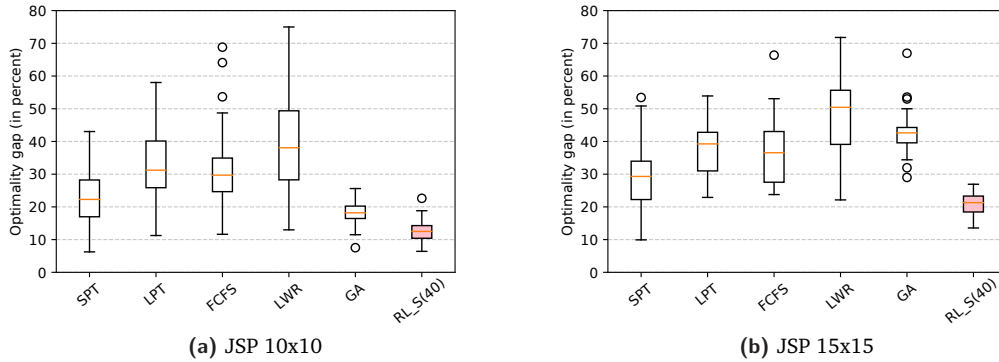


Fig. 4.17.: Comparison of the distance to the optimal solution in the *classic* JSP ($\lambda = 0$) between: different heuristics, a metaheuristic GA and our RL approach with sampling applied.

the Transformer encoder becomes a very memory consuming part on the model. The memory demanded is so large we could not evaluate the model for instances larger than JSP10x10. Hence, this alternative is not further considered.

Fig. 4.17 shows the optimality gap for the instances in which the optimal solution can be obtained in a reasonable time: JSP10x10 and JSP15x15. The optimality gap is defined as the difference (in percent) between the solution obtained and the optimum. According to the results, the RL_S(40) outperforms the rest of the heuristics and metaheuristics. However, it presents a difference in performance when compare to the solver that goes from a 11.2% in the JSP10x10 up to 27.5% in the JSP25x25 (see Tab. 4.2).

Despite the solver performs better than the RL model, the time required in each case are totally different, and thus, carrying out a fair comparison is tricky. For this reason, we perform a comparison taking into account the time required to compute the solution in both cases. We observe in the classical JSP ($\lambda = 0$), the solver outperforms the other alternatives, and it does in a competitive computational time. Nevertheless, the situation is reversed when the *limited idle-time* variant is considered. Despite this problem adds no much complexity, the computation time required by OR-Tools increases significantly. In this case, a slight increase in the number of

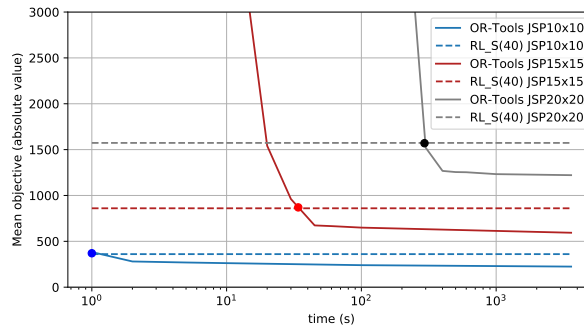


Fig. 4.18.: Mean objective value in function of the time obtained by OR-Tools for the JSP with limited idle time ($\lambda = 1$). The performance of the RL model is depicted in dashed line. The intersections between both representations are highlighted to indicate the time required for the solver to match the RL model.

constraints in the problem is enough to prevent the solver from getting good approximations in the short time. For this reason, we aim to calculate the time required by the solver to achieve a solution comparable with that of the RL model. The results obtained are depicted in Fig. 4.18. In essence, for problem instances larger than JSP15x15, the time required by the solver to match the performance of the RL model is orders of magnitude higher than the inference time. This shows that although the solver ends up achieving better results, the time they require to operate is non-comparable.

Hence, several conclusions can be extracted from this experiment. Firstly, we observe that for small size instances of the problem, the CP solver achieves a close to optimal solution in a short period of time (comparable to the inference time). In that case, the CP solver behaves better than the heuristics and the NCO model. However, when the size of the problem increases the solver is not a viable solution as it requires exponentially larger response times to obtain competitive solutions. In that case, heuristics represent a better option. Particularly, **for low to medium size instances, the NCO model is capable of inferring a better heuristic than the classical references studied.** In this range, the NCO model represents the best alternative. We find this optimal spot around the instances JSP10x10-15x15 in the problem. **However, or larger sizes this advantage fades and the policy performs similar to the classical heuristics.** The summary of the competitiveness of the different strategies is depicted in Fig. 4.19.

Finally, regarding the idle-time constrained variant or other complex versions of the problem it is hard to find heuristics that specifically suits the definition, thus in these scenarios NCO presents an major advantage over the alternatives.

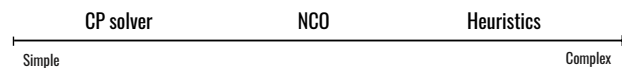


Fig. 4.19.: Competitiveness of the different OR strategies for addressing CO problems in close to real-time. NCO represents the best strategy in the low to medium range of the problems, where it achieves a better behaviour than classical heuristics.

4.5 Conclusions

In this Chapter, the learning process in a toy CO problem is analyzed. Several experiments are conducted in this problem to draw conclusions applicable in the experimentation. In this scenario, an optimality gap is observed in the model performance. Even in this environment where the combinatorial space is small and the number of instances that form the input combinations is limited, the model does not achieve the optima solution for all cases. After performing a supervised learning using the optimal labels, the model perfectly overfits the whole space. Then, we conclude that the neural agent is complex enough to embed the optimal behavior, however the learning algorithm is not extracting enough information from the problem. In order to improve the policy that the agent infers, different strategies are evaluated: a numerical optimization is used to determine the convergence achieved, entropy regularization is used to extend the learning period and PPO technique is used to avoid degradation in the policy previously experienced in vanilla PG, allowing a better convergence on the model. After applying these techniques we manage to reduce the optimality gap, however the error still exists. We empirically determine that for RL it is asymptotically difficult to update the policy in the right direction as the policy gets closer to the optimal behaviour. The closer to the optima is the model, the weaker the impulses to improve the policy are. However, this drawback does not affect the purpose of this Thesis, as optimal solutions are not required. Instead good enough solutions in a reasonable time are sought.

Also, after testing different neural architectures and RL techniques, we conclude that a custom fully-vectorized implementation of the model would significantly reduce the learning time. In this sense, this fully-vectorized implementation allows us to perform a new learning method that benefits from the features of our implementation. We name this method a *self-competing strategy*, and it is able to estimate a state-dependent baseline $b(s)$ without requiring an additional estimator or critic, just creating a reward distribution from the policy itself. This method comes at almost no time cost, as the operations required are executed in parallel.

Finally, the proposed learning framework is tested in a classical combinatorial problem, the Job-shop Scheduling problem (JSP) and the constrained no *idle-time* variant. Conducted experiments prove that there exists a range of problem sizes where the proposed solution is superior for computing rapid solutions when compared to classical heuristic, metaheuristic, and Constraint Programming (CP) solvers. Particularly, we observe that **the recurrent NCO model in the low to medium range of problem sizes** infers a policy that outperforms the classical handwritten heuristics. However, for larger problem sizes this advantage fades and the policy performs similar to classical algorithms. For other variants as the JSP without idle-time constraints or other complex versions of this problem it is hard to find heuristics that specifically suit the definition, thus in these cases NCO presents a major alternative. On the other hand, we also conclude that the attentional model is not a viable alternative in this scenario. Due to the large state representation required in this problem, the memory the model demands is excessive.

Use-case: Application for 5G real-time placement decision systems

” *"In God we trust. All others must bring data."*

— W. Edwards Deming

Contents

5.1	Introduction to Network Function Virtualization	72
5.1.1	Benefits of Network Function Virtualization	73
5.1.2	Description of the ETSI-NFV architecture	73
5.1.3	Network service creation process	74
5.2	VNF Placement Optimization	75
5.2.1	Related work on the VNF Placement problem	76
5.2.2	VNF Placement problem formalization	77
5.3	Experimentation details	80
5.3.1	Model implementation	81
5.3.2	Performance comparison	84
5.3.3	Results	85
5.3.4	Learning and inference times	88
5.4	Conclusions	89

THE main objective of the 5G technology is to provide a unified infrastructure able to meet the growing demand of services with heterogeneous requirements that currently the telecommunication network is beginning to experience. According to ITU-R, 5G is envisioned to provide: enhanced mobile broadband (eMBB), massive machine-type communications (mMTC), and ultra-reliable low-latency communications (URLLC). The next iteration of the mobile network is required to cost-effectively embrace those services of different nature. To fulfill these requirements, and as it will be discussed in this Chapter, network virtualization technologies would be of utmost importance.

5G is called to introduce a major transformation in communication networks within its transition into cloud native architectures. The next iteration of the mobile network leverages on Network Function Virtualization (NFV) and Software Defined Networks (SDN) technologies to introduce unique network capabilities that will drive innovative applications. In this framework, traditional network functions deployed over dedicated equipment are evolved to software implementations that run on general-purpose hardware.

One of the main challenges of deploying NFV is the optimal resource placement within the infrastructure. The Virtual Network Function (VNF) placement and network embedding can be formulated as a mathematical optimization problem concerned with a set of constraints

that express the restrictions on the network infrastructure and the service level agreements (SLA). **The development of real-time decision-making systems enables to compute rapid approximations to that combinatorial problem would be of utmost importance to optimize the network usage**, and thereby, to contribute to the implantation of NFV in the next generation mobile network.

5.1 Introduction to Network Function Virtualization

Traditionally, telecommunication networks were based on proprietary 'hard-wired' equipment. However, this reliance on physical infrastructure presents many drawbacks: deployments have high capital & operational expenditures, but also updating functionality cannot be done in a simple way. In addition, hardware lifecycles are becoming shorter as technology innovation accelerates, therefore, this reliance on dedicated equipment is inhibiting the roll out of new network services and it is constraining innovation. As a result, Network Function Virtualization (NFV) has emerged from the industry, promising to transform the way that network operators design, manage and deploy network infrastructure thanks to the current advances in virtualization technologies.

In this paradigm, virtualization is used to decouple physical network equipment from the functions that run on them. Hence, NFV offloads network functions to software implementations that run on top industry-standard hardware. Virtualization provides the modularity and isolation for each function, so that they can operate independently inside a general purpose environment. This results in a more efficient, scalable, and automated network infrastructure. The adoption of this technology gives also to Mobile Network Operators (MNO) flexibility in the life-cycle management of the services (e.g., creation, deletion, horizontal or vertical scaling operations), and also enables them to relocate, scale and instantiate VNFs at different physical network locations without requiring hardware modifications. Services may be located in datacentres, network nodes or even in the end user premises in a seamless manner.

As mentioned, NFV is a paradigm that facilitates the dynamic provisioning of network services (NSs). In this architecture NSs are created connecting or chaining together individual VNFs through the infrastructure. Services are orchestrated as a whole entity, and resources dedicated to them are adapted to meet the fluctuations of the traffic. This architecture guarantees therefore an efficient provisioning of the services and meet the Service Level Agreements (SLA), while keeps the operational costs low thanks to the reutilization and sharing of the resources.

Network Functions Virtualization (NFV) is highly complementary to Software Defined Networking (SDN), but not dependent on it. SDN complements NFV by offering programmatic access to the abstracted network resources and full programmability of forwarding capabilities. SDN control capabilities are used to implement dynamic traffic steering policies so that flows are dynamically routed along a path traversing the VNF instances composing a given network service. NFV and SDN technologies together introduce a level of flexibility in network service provisioning key for coping with requirements of the complex and unpredictable traffic patterns in modern networking system.

In this context, an appropriate orchestration mechanism is required to support such operational flexibility and gain responsiveness. This must be done in order to guarantee the target operating margins. Therefore, orchestration mechanisms should account for both business value and customer experience respectively: (i) cost-effectively controlling the resource utilization, to

achieve the target range of operating margins; and (ii) fulfilling of Quality of Service (QoS) objectives specified in the Service Level Agreement (SLA) between a customer and a service provider, which are typically expressed as technical performance metrics.

5.1.1 Benefits of Network Function Virtualization

NFV presents an opportunity, through the flexibility afforded by software appliances operating in an open and standardised infrastructure, to rapidly align management and orchestration to well defined standards. The application of NFV brings many benefits to network operators, contributing to a dramatic change in the telecommunications industry [121]. The benefits of this architecture are summarized as follows:

- **Reduced equipment costs** through consolidating equipment and exploiting the economies of scale of the IT industry. As it enables a wide variety of ecosystems and encourages openness.
- **Accelerated Time-to-Market.** NFV reduces the time required to deploy new services. In addition, it lowers the risks associated with rolling out the market opportunities.
- **Target service based on geography or customer-based.** Service velocity is improved by provisioning remotely in software without requiring hardware installation. In addition, services can be scaled up/down as required in specific locations, even close to the use (Multi-Access Edge Computing — MEC).
- **Optimizing network configuration and topology** in near real time based on the actual traffic patterns and service demand. For example, optimization of the location and assignment of resources to network functions automatically and in near real time could provide resiliency to changes in the network.
- **Supporting multi-tenancy** thereby allowing network operators to provide tailored services and connectivity for multiple users, all co-existing on the same hardware with appropriate secure separation of administrative domains.
- **Reduced energy consumption** by exploiting power management features in standard servers and storage, as well as workload consolidation and location optimization.

5.1.2 Description of the ETSI-NFV architecture

In order to standardize the NFV technology, in 2012, seven of the world's leading telecom network operators created a specification group, the European Telecommunication Standardization Institute (ETSI) Industry Specification Group (ISG) on Network Functions Virtualization (ETSI ISG NFV). As a result of this engagement, currently major Mobile Network Operators (MNO) are involved in the development of this technology.

So far, the ETSI ISG NFV has developed over 100 different reports and specifications for the virtualization of network functions. NFV publications describe and specify virtualization requirements, architecture framework, functional components and their interfaces, as well as the protocols and the APIs for these interfaces. In addition, the ISG NFV also studies VNF performance, reliability, resiliency and security challenges linked to virtualization.

The ETSI-NFV reference architecture¹, which is depicted in Fig. 5.1, consists of three main elements:

¹The reference NFV architecture is defined in the standard ETSI GS NFV 002.

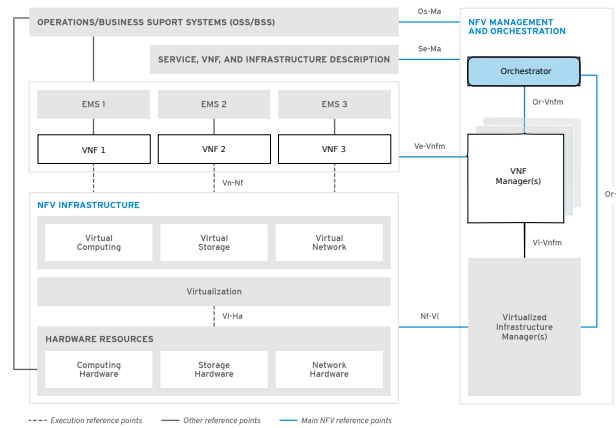


Fig. 5.1.: ETSI-NFV reference architecture. The different components that form the architecture are depicted together with the standardized interfaces that interconnect them.

- **Virtualized Network Functions (VNFs)** are the software implementations of the network functions that are deployed on a network virtualization infrastructure. As described, VNFs can be interconnected to compose a Network Service (NS).
- **NFV Infrastructure (NFVI)** is the hardware and virtualization software that build the environment where VNFs are deployed. The NFVI can be distributed in several Points of Presence (NFVI-PoP).
- **NFV Management and Orchestration (NFV MANO)** is the architectural framework that covers the orchestration and life-cycle management of physical and software resources that support the infrastructure virtualization and the deployed services.

NFV Management and Orchestration

The NFV Management and Orchestration (NFV MANO) is an architectural framework that coordinates network resources and the allocated services within the infrastructure. As such, the automatization that it provides is crucial for ensuring rapid, reliable network operation at scale. An essential characteristic for an effective orchestrator is to facilitate the onboarding of the NSs. With the NFV MANO architecture, this is straightforward and can be performed by the service provider without the involvement of the network supplier. Other key challenge MANO addresses are operational issues to enable scalability and security in multitenant scenarios.

The NFV MANO architecture consists on three major functional blocks:

- The **Virtual Infrastructure Manager (VIM)**, responsible of controlling and managing the NFVI compute, storage, and network resources within the operator's infrastructure domain.
- The **VNF Manager (VNFM)** is the entity in charge of the life-cycle management of the VNFs, from deployment to termination, keeping track of their status to adjust their configuration if needed.
- The **NFV Orchestrator (NFVO)** is the entity in charge of Network Service (NS) life-cycle management: creation, termination, monitoring, scaling, etc., via coordination between the NFV MANO elements, such as the VIM and VNFM.

5.1.3 Network service creation process

The adoption of virtualization technologies in networking is promoting a radical innovation in the way network services are delivered. In the NFV framework, a virtualized Network Service

(NS) is created interconnecting individual VNFs. The composition of the ordered set of VNFs to be deployed in order to provide a service is referred to as a Forward Graph.

To deploy a NS on the NFVI, the following steps are involved. In this procedure, the role of the NFVO is to automate the creation process. The orchestrator generates instructions to control the VIM. That is, the NFVO receives a network service descriptor together with the instantiation parameters and transform it into a deployable template² the VIM uses to create the underlying VNFs, virtual networks and virtual links.

During the NS creation, the underlying virtual machines (VM) are instantiated and the network configuration occurs. In this process the connections between VNFs are created. Based on the VNF Forward Graph Descriptor (VNFFGD) included in the NS Descriptor (NSD), the VIM communicates the SDN controller the flows to create between VNFs. However, the service function chaining (SFC) instead of relying on conventional routing methods that route the packets using destination IP address, it creates traffic flows that emulate a series of physical network devices with cables linking them together inside the infrastructure. Being during runtime that the SDN Controller establishes the path for the data packets. In simple words, before departing from a port the data packet asks the SDN controller about the destination and the controller determines it. Indeed, SDN control capabilities may be used to implement dynamic traffic steering policies so that flows are dynamically routed along a path traversing the VNF instances composing a given network service.

5.2 VNF Placement Optimization

NFV is not limited to datacentres, but also the network nodes or even the end user premises can be used as part of this infrastructure. In particular, Multi-access Edge Computing (MEC) is the technology that allows to extend cloud computing capabilities to the edge of cellular networks. Thereby, MEC provides storage and computational resources all the way to the edge. This allows to the operator to distribute services, and in addition enables application developers and content providers to place services closer to the end-user.

In this scenario, where network services can be distributed along the telecommunication network, even in the end-user premises, the optimization of service deployment is one of the most challenging tasks in orchestration. Optimizing the placement of services is of utmost importance to fulfill the efficient utilization of the expected virtualized network. In this sense, **this Thesis seeks to build a real-time decision-making system that interacting with the NFVO could orchestrate intelligent deployments.**

Specifically the challenge addressed in this Chapter is known as the VNF Placement problem and is one of the core NFV Resource Allocation problems, together with the VNF Chaining Composition and VNF Scheduling problems. The VNF placement addresses a key problem in dynamic VNF chain provisioning in a cloud environments. It consists on efficiently mapping a set of network service requests on top of the physical network infrastructure. Particularly, it seeks to obtain the optimal placement for a NS chain considering the state of the virtual environment, such that a specific resource objective is accomplished (e.g., maximization of the remaining resources, minimization of the overall power consumption, optimization of a specific QoS metric, etc.). Besides, specific aspects of NFV, such as forwarding latency, ingress/egress bitrate and flow chaining, have to be taken into account.

²In the ETSI-NFV standard, the service templates follow the TOSCA Oasis specification. The service template describes what is needed to be preserved across deployments to enable interoperability between cloud providers.

Tab. 5.1.: Update frequency at the different stages of the mobile telecommunication network.

Domain	Parameter type	Network entities	Update frequency
Network design	Deployment parameters	RAN, Core	Montly
Network optimization	Network parameters	Cells	Weekly/Hourly
RAN optimization	L1 to L3 parameters	Radio	Seconds

The VNF placement can be formalized as a CO problem, a constrained maximization or minimization program in which the constraints model natural or imposed restrictions on the deployment. In this case, constraints are imposed by the network infrastructure and the Service Level Agreements negotiated for the services to be allocated. This problem has been expressed as NP-hard; therefore, it is only tractable to exactly solve it for small instances.

In the envisioned scenario, this problem has to be solved for every new service deployment. Service petitions are required to be executed within a limited time. Due to this reason, fast approximate solutions are expected to the problem. However, the network definition does not vary so often in the infrastructure, it takes days or even weeks between network optimization events occur (see Tab. 5.1). One can benefit from this and build a model adapted to the specific network conditions at the moment. In this regard, updates on the model are required at every change in the network infrastructure. Namely, a new learning process has to be performed to adapt the model to the new conditions on the interval.

5.2.1 Related work on the VNF Placement problem

The VNF Placement problem has raised a considerable interest in the research community. Several previous works in the literature have undertake this problem, e.g. [3, 13] provide a good definition of the problem. As it is an NP-hard optimization problem, run-times to optimally solve it are unaffordable for large instances. For this reason, this problem has been tackled by applying the following alternatives:

- Optimal solutions has been achieved for small problem instances. For example, using MIP [82] [48] or analytically solving the Bellman equation on the problem described as a Markov Decision Process (MDP)[101].
- Another alternative is the heuristic approach that, despite not guaranteeing convergence to local optima, is able to obtain competitive solutions in reasonable computational times. For instance, see the work in [92] and [30].
- Finally, as an extension to heuristic approaches, metaheuristics provide a high-level problem-independent algorithmic framework that sets the guidelines to develop optimization algorithms. These approaches find near-optimal solutions by iteratively improving intermediate solutions with regard to a given measure of quality [76].

Based on the previous alternatives, different objectives have been addressed when optimizing the VFN Placement. Minimizing the number of virtual network function instances mapped on top of the infrastructure [69], or maximizing the number of successfully embedded service requests [92] are a couple of these objectives. Other works deal with power-based placement proposals. For example, minimizing the power consumption via a genetic algorithm approach [62] or providing robustness to unknown or imprecisely formulated resource demand variations [74], are some of these works.

Solving the VNF Placement problem with Reinforcement Learning

In addition to the previous references, ML has become a promising research line for solving problems of this nature. In particular, the recent idea of learning placement heuristic without human intervention has gain attraction in the community. To that end, Reinforcement Learning has been used in the literature. This technique learns through interaction with the problem how to exploit the problem characteristics to infer placement decisions. Is the case for example of [77], that uses Q-learning to control the resources allocated as part of the NFV management system. In [78], the author also employs a Deep Neural Network (DNN) to make resource reallocation decisions based on his previous work. [126] utilize for the first time, historical network request data and policy-based RL to optimize node mapping. They use an embedding representation of the node and links which, at each time step, are updated with changing features of the network attributes. This network embedding is passed through a Convolutional Neural Network (CNN) to select the substrate node that maximizes the long-term revenue. However, these works present a major concern. They do not include a comparison on the results against other resolution methods, in essence, they lack of an optimality study.

5.2.2 VNF Placement problem formalization

This Thesis follows this novel research line of using NCO to obtain the optimal workload allocations. Particularly, the objective pursued is the minimization of the power consumption of the infrastructure. With the global climate change leading our concerns these days, the power consumption of datacenters has become a key issue. Also, the main provisioning cost of VNF instances is due to power consumption to operate servers and cooling facilities. Which is largely decided by the numbers and the types of VMs running different VNFs. Therefore, the optimization of the power consumption that this Thesis pursues is a goal that not only benefits to the infrastructure providers but also the environment.

Nonetheless, there is not a single definition of the VNF Placement problem throughout the literature. And the way in that the problem is defined is key for the selection of the resolution method. In the more general case, the problem can be seen as a graph allocation problem. In that case, the network service can be defined as a directional graph in which each node represents a VNF instance and the edges, the traffic flow between entities. In the same way, the infrastructure can be also represented by a graph. In this occasion, the nodes stand for the computing hosts and the edges for the interconnect capabilities. However, addressing this graph allocation problem using ML is not trivial. In the literature mentioned above, several assumptions are done on the resolution. Mainly, the deep learning models used are limited to fixed input sizes, narrowing their applicability to real scenarios; they heavily rely on other algorithms to reinterpretate the output of the model, or even parts of the solution are computed externally to the model. Due to this reasons, we look for a trade-off between the definition of the problem to solve, the applicability of the solution on real scenarios and the quality on the solutions obtained.

This Thesis is focused on obtaining a model that by its own computes a complete solution to the problem. Therefore, this problem is formulated using NCO approach that recently has obtained outstanding results addressing these problem. To this end, **two main assumptions are made, instead of addressing the problem as a graph problem, it has been loosen to a sequence problem. This implies that the service are treated in its simplest form, which correspond to a linear sequence of VNFs and the infrastructure is reduced to a start topology to not deal directly with path in addition to node selection. We refer to this problem as the VNF chain embedding problem or VNF-CE.**

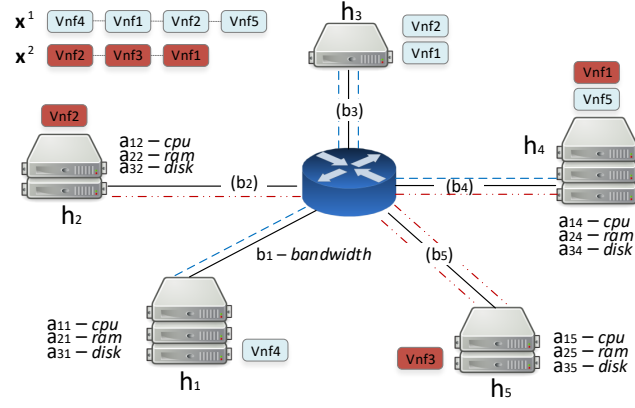


Fig. 5.2.: Example of network service allocation over a virtualized environment. It depicts an environment in which the service chains, denoted as x^1 and x^2 , are optimally placed in a total of five hosts (from h_1 to h_5). Each host has its own capacities for a_{1h} computation, a_{2h} memory and a_{3h} disk capacities. Regarding the network, each host is connected to a common switch through a dedicated link i , associated to its capacity bound b_i .

Mathematical formalization of the VNF-CE problem

Let us start with the mathematical formalization of the VNF-CE problem. Consider a list of network services that have to be optimally placed in a set of host servers $h \in H$, and each of those host servers relies on a limited amount of available resources $r \in R$, in terms of computing, storage and connection capabilities. As previously mentioned, the host servers are interconnected through a star topology using their own link connections $i \in L$ (see Fig. 5.2). Link attributes as bandwidth or propagation delay are also considered. **The final purpose of this problem is to discover the optimal placement for a given service chain that minimizes the total power consumption of the infrastructure. This solution is subject to complying with the restrictions associated to the availability of virtual resources and link capacities, as well as the latency thresholds that each service imposes.**

In order to formulate the problem, we take the nomenclature proposed by (Marotta et al., 2017) [74]. Let us denote as $\{h_1, h_2, \dots, h_n\}$ the set of host servers H , and let V be the set of VNFs available in the repository. So, a network service consists of an array of $m \in \{1, \dots, M\}$ virtual network functions that compose a service chain $\mathbf{x} = (x_1, x_2, \dots, x_m)$. Each element x in the service corresponds to a VNF $v \in V$, and the combinatorial space of all services is denoted as \mathcal{X} .

The problem consists of finding the optimal set of placements denoted as $\mathbf{w} \in \{0, 1\}^{m \times n}$, where w_{xh} stands for a boolean status variable that describes whether the function x is placed in host $h \in H$ or not (1 in the positive case, and 0 in the negative). Then, the search space in which the solution for the problem needs to be found is $\Omega = \{\mathbf{w} \in \{0, 1\}^{m \times n} \text{ s.t. } \sum_h w_{xh} = 1 \forall x\}$. The restriction in Ω states that a function can only be placed in one host at a time.

Prior to presenting the problem cost and restriction functions, a summary of the decision variables and parameters of the problem is presented in Table 5.2. As stated previously, the set of variables to optimize are those that define the placement \mathbf{w} . To support the problem description, auxiliary variables are presented. This is the case of the server activation variables $k_h \in \{0, 1\}$, which indicate 1 if the server is executing any VNF, and powered off otherwise; and link activation variables $g_i \in \{0, 1\}$, which are equal to 1 if the link i is carrying traffic and 0 otherwise.

Tab. 5.2.: Problem formalization variables.

H	set of hosts
L	set of links
V	set of VNFs
S	set of NS chains
R	set of resources
P	set of placements
a_{rh}	amount of resources r available in host h
r_{rv}	amount of resources r requested by VNF v
W_h^{min}	idle power consumption of host h
W_h^{cpu}	power consumption of each cpu in host h
W_{net}	power consumption per bandwidth unit on links
b_i	bandwidth of the link i
b_v	bandwidth demanded by VNF v
c_v	computation time of VNF v
l_i	latency on the link i
l^x	maximum latency allowed on the service chain x
w_{xh}	binary placement variable for function x in host h
k_h	binary activation variable for host h
g_i	binary activation variable for link i

In relation to the power consumption, the host servers are characterized by a linear power profile that grows in proportion to the computing utilization. Each server activated ($k_i = 1$) consumes a minimum power W_h^{min} , and its power increases with the sum of the CPU demanded by the VNFs assigned to the server. Each cpu in use consumes W_h^{cpu} watts. Regarding links, they also have an energy cost associated. This is calculated multiplying the cost per bandwidth utilization, denoted as W_{net} , by the bandwidth utilized in each link. The available resources, $r \in R$, that each server h owns are denoted as a_{rh} . The amount of resources r needed by VNF v is indicated in r_{rv} . With regard to the network connection, if consecutive VNFs in the chain are placed in the same server h , they are internally interconnected and therefore there is no link usage. **This networking condition makes this problem non-linear.** In this sense, the bandwidth occupied for data transfer of the element x related to the VNF v and expressed as b_x , is 0 if placed in a same host as the previous one or b_v otherwise. In the same way, l_x represents the latency the element x generates in the link i . And c_v the computational time required by VNF v . Finally, the maximum bandwidth allowed in link i is represented as b_i . And the maximum latency allowed for each service chain x is denoted as l^x .

$$b_x = \begin{cases} b_v, & \text{if } p_{x-1} \neq p_x \\ 0, & \text{otherwise} \end{cases} \quad l_x = \begin{cases} l_i, & \text{if } p_{x-1} \neq p_x \\ 0, & \text{otherwise} \end{cases}$$

Tab. 5.3.: VNF-CE problem equations.

$$\arg \min_{\mathbf{w} \in \Omega} \left(\sum_h [W_h^{cpu} \cdot \sum_x (w_{xh} \cdot rrv) + W_h^{min} \cdot k_h] + \sum_i W_{net} \cdot \sum_x (w_{xh} \cdot b_x) \right) \quad (5.1)$$

s.t.

$$\sum_x w_{xh} \cdot rrv \leq k_h \cdot a_{rh} \quad \forall h \in H, r \in R \quad (5.2)$$

$$\sum_x w_{xh} \cdot b_x \leq g_i \cdot b_i \quad \forall i \in L \quad (5.3)$$

$$\sum_h \sum_x c_v \cdot w_{xh} + \sum_i \sum_x l_x \cdot w_{xh} \leq l^{\mathbf{x}} \quad \forall h \in H, i \in L \quad (5.4)$$

The cost function to optimize is presented in Eq. (5.1). It represents the power consumption, and it is calculated as the sum of the power consumption related to the activated servers and the aggregated cost of the active links. The constraint in Eq (5.2) determines that the total resources used in a server must not exceed the available ones a_{rh} in active servers. Therefore, it sets a link between server activation variables k_h and allocation variables w_{xh} ; only the servers that host some VNF are active. Then, the capacity constraints for bandwidth are defined in Eq. (5.3). It uses link status variables g_i in the same way as host activation ones. They link the boolean status of links to the status of the node activation variables: if a link is used, then its end-nodes must be activated; if a node is not activated, then neither is the associated link. Finally, the constraints in Eq. (5.4) express the latency requirement for a network service \mathbf{x} , establishing that the aggregation of the latency over the links used in the networking and the latency due to computation time must meet the latency limit $l^{\mathbf{x}}$ for the service.

For the sake of illustrating the problem, an example is depicted in Fig. 5.2. Here, an equivalent representation for the placement is introduced: $\mathbf{p}^{\mathbf{x}} = (p_1, p_2, \dots, p_m)$ where $p \in H$. This notation will be especially useful for simplifying the formulation of the RL equations that will be described hereafter. To continue, each server is connected to a common switch through a dedicated link i . The objective is to place the service chains (denoted as \mathbf{x}^1 and \mathbf{x}^2) minimizing the overall cost function. In this example, the placement vectors computed for each service are $\mathbf{p}^{\mathbf{x}^1} = (h_1, h_3, h_3, h_4)$ and $\mathbf{p}^{\mathbf{x}^2} = (h_2, h_5, h_4)$.

5.3 Experimentation details

In the experimentation three different environments with 10, 20 and 50 host servers are tested. Resources in those environments are initially occupied following a **uniform distribution**. In these environments, a service chain of $m = \{5, 7, 9, 11\}$ elements is required to be allocated minimizing Eq. 5.1. The VNFs that conform the chain are chosen from a dictionary V of 10, 20 and 50 elements respectively. We refer to these problems as VNF-CE10, VNF-CE20 and VNF-CE50.

The description of the hosts and the VNF Dictionary used as a reference to build the evaluation environments are the following:

Host Types		VNF Types	
<i>No. CPUs</i>	[4, 6, 8, 12, 16]	<i>No. CPUs required</i>	[1, 2, 4, 6]
<i>Link BW (Mbps)</i>	[400, 600, 800, 1000]	<i>Bw required (Mbps)</i>	[10, 20, 40, 60, 100]
<i>Link Latency (ms)</i>	[5, 10, 15, 20, 25]	<i>Processing latency (ms)</i>	[10, 20, 40, 60, 100]

To address this problem we evaluate two alternatives based on the Markovian model introduced in Section 3.4.2. The first model (a) uses RNNs to encode the service request, whereas the second one (b) is based on a fully-attentional architecture. The operations in both cases behaves as follows, a network service \mathbf{x} formed by a sequence of VNFs, each one represented by its specific features, is embedded and encoded. This step differs depending on the model used. In (a) the encoding process is recurrent i.e., the encoder iteratively operates over the input; whereas in (b), the Transformer encoder directly access the input representation to encode the input information with the relative position on the chain. The resulting vector represents the static part of out input, and it is combined with the state of the environment d_t to feed the FFN that iteratively decides the server in which each VNF in the chain is going to be located.

In this problem, due to the complex relations between the constraints for physical resources (i.e., server capabilities [Eq. 5.2] and link capabilities [Eq. 5.3]), it is hard to address them using a masking scheme. Also, the restrictions associated to the whole service (i.e., the end-to-end service latency [Eq. 5.3]) cannot be checked until the complete solution is computed. Therefore, all these constraints are relaxed and introduced as penalty terms into the objective function.

The parameters related to power consumption in the environment are the following:

$$\begin{aligned}
 W_h^{min} & 200 \text{ (watt)} \\
 W_h^{cpu} & 100 \text{ (watt)} \\
 W_{net} & 0.1 \text{ (watt / mbps)}
 \end{aligned}$$

Lastly, the Lagrange multipliers hand-selected to penalize the different constraint functions are presented below:

$$\begin{aligned}
 \lambda_{occupancy} & 1 \\
 \lambda_{bandwidth} & 0.01 \\
 \lambda_{latency} & 0.01
 \end{aligned}$$

5.3.1 Model implementation

The neural architecture used in the VNF-CE problem presents some peculiarities. In this problem, the model operates on a single sequence that represents a single service request at a time. The procedure iterates placing at each step t each element in the chain, until the whole service is completed. Therefore, the number of decisions to be made is equal to the number of VNFs in the service m . This direct relation between the input and the output enables to train a model that performs specially well on this problem.

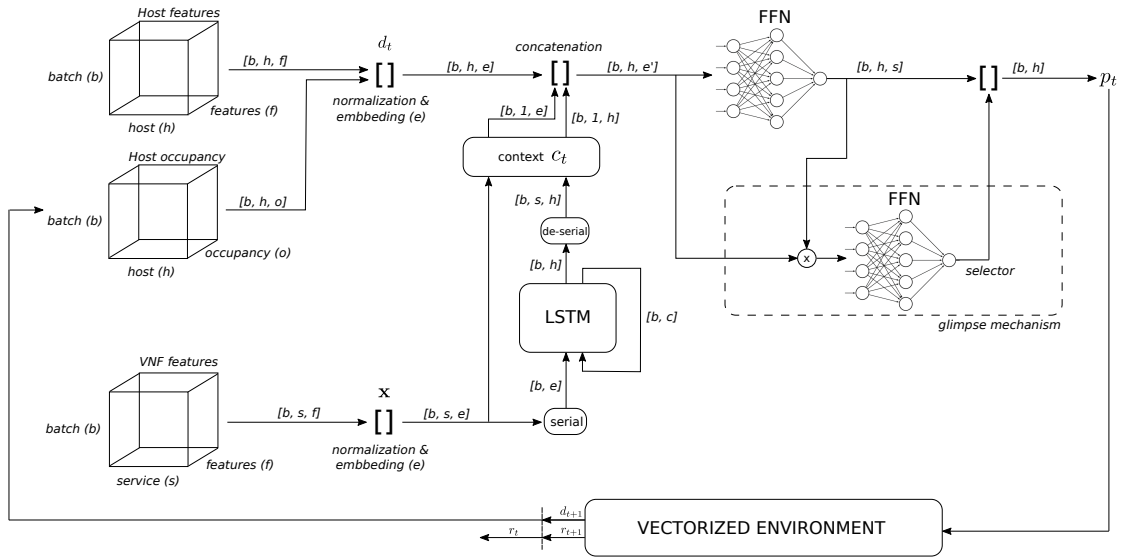


Fig. 5.3.: Recurrent neural agent for solving the VNF-CE problem. The service chain x represents the static part of the model, as the features of the service do not change during the problem resolution. These features are sequentially serialized and encoded using a RNN. The result is concatenated through the context vector c_t with the host environment d_t , which dynamically changes its features at each VNF is placed on the infrastructure. Finally a memory-less FFN is in charge of computing the policy distribution over the actions.

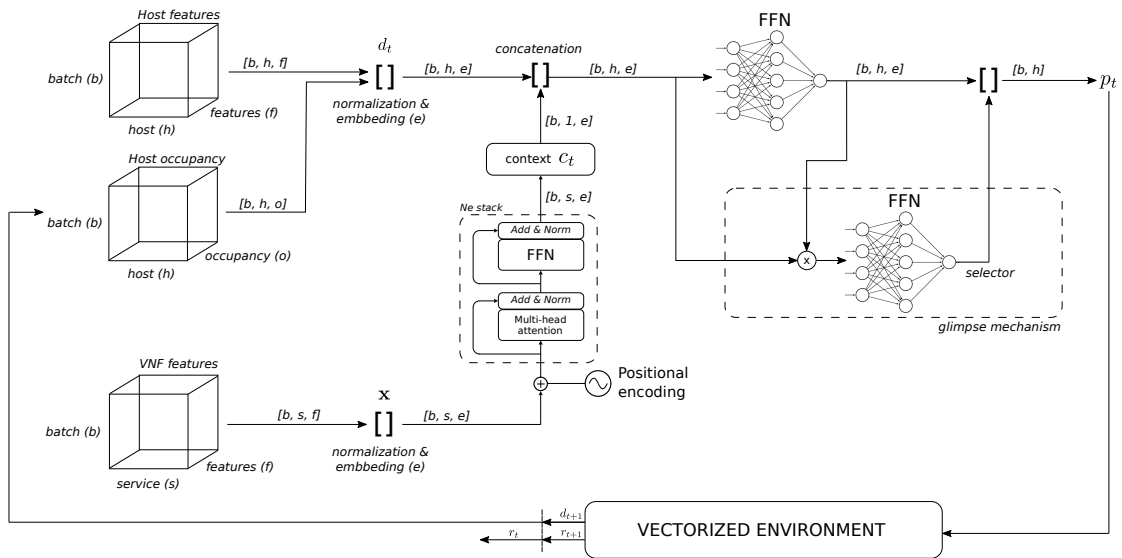


Fig. 5.4.: Fully-attentional agent for solving the VNF-CE problem. The service chain x is self-attended using a multi-headed Transformer encoder. In doing so, the decoder receives a direct representation on the input sequence without using recursion. The attention vector is concatenated with the host environment d_t , which dynamically changes its features at each VNF is placed on the infrastructure, to create the context vector c_t . Vector from which a memory-less FFN decoder computes the policy distribution over the actions.

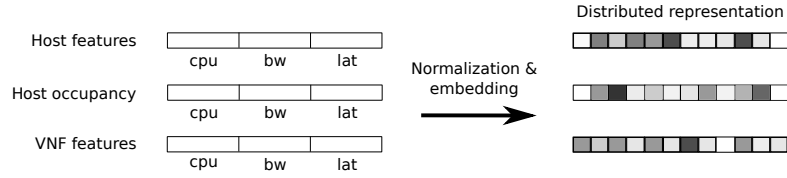


Fig. 5.5.: The state of the problem is gathered in three tensors: the host feature tensor, the host occupancy tensor and the VNF feature tensor. These features are normalized and embedded to provide a representation that enriches the feature extraction on the model.

The complete description of the model, highlighting the dimensionality of the tensors, is depicted in Fig. 5.3 for (a) the recurrent model and in Fig. 5.4 for (b) the fully-attentional model. In both cases three tensors are used to define the state of the problem: the host feature tensor, the host occupancy tensor and the VNF feature tensor (see in Fig. 5.5). The host features tensor defines the infrastructure, the maximum capacity of the host and links that conform the network. The occupancy tensor, the usage of the physical resources. And finally the VNF feature tensor, the service characteristics. In the two first cases, the tensors are merged to define the state of the infrastructure and create dynamic occupancy tensor d_t . The last VNF feature tensor x is configured at the beginning of the resolution process and it maintains static. The procedure operates as follows, in (a) the service x is serialized and encoded backwards, thus the representation of the remaining elements to be placed in the infrastructure at every step is obtained. For every element x_j in the chain we refer to its encoded vector as $e_j = enc(x_j, \dots, x_m) \forall j$. This procedure is computed once and stored to be used during the resolution of the instance. Alternatively, in (b) computing the feature vector is not required as at every step in the decoding process the model attends directly over the input. As mentioned, the iteration with the problem has a fixed number of steps. In this sense, the *context* mechanism c_t that operates over the encoded sequence, works synchronously with the decoding mechanism. The context vector c_t increments iteratively pointing at the element in the chain to be placed i.e., it points at the same position over the input, the decoder is working on the output. At every step t , the context is created concatenating the VNF to be placed x_j together with the features of the remaining chain e_j . Likewise, it is joined to the state of the problem d_t to create the input for the decoder ($s_t \doteq \{c_t, d_t\}, t = 1, \dots, m$).

The decoder is formed of a memory-less FFN supported by a glimpse mechanism attached to the state representation. Here, the information of the occupancy on the hosts d_t is used to introduce this key information deeper into the model. In particular, it is used to support the final selection for the host.

Finally, it is important to remark that the model is completely implemented over tensor operations. This enables to execute the complete model on the GPU unit and significantly accelerate the computation of the learning process. To this end, the problem description has to be specifically implemented as tensor operations. This cannot be done a directly using linear operations, but conditional operations has to be included to represent the non-linearities. This is possible in the case of using PyTorch, as this framework interprets the tensor operations in execution time.

Hyperparameters

This section provides the details on the neural model particularized for the VNF-CE. With regard to (a), given the relatively small length of the input sequences $m = \{5, 7, 9, 11\}$, the LSTM encoder only requires of a single layer of hidden size 16 to code the information of the NS. On the other hand, the Transformer encoder is formed by $N_e = 2$ stacked self-attentional sub-layers. Each of them with 2 heads in the multi-headed product-based attention layer. The complexity of

the neural model needs to be sufficiently large to embed the features of the problem. Yet, once this requirement is satisfied, increasing its complexity does not result in better performance. Prior to the encoding, both sources are normalized and embedded single linear layers with a vector size of 64 in both cases. Normalizing the input vectors and embedding them in a higher feature space yields to superior solutions.

In regard with the decoder, the DNN used consists on multiple dense layers with a ReLU activation. The variables are initialized with Xavier initialization [46]. The batch size is 800, and it is formed by 20 different instances introduced 40 times each, following the self-competing baseline introduced in Sec. 4.2.4. The optimizer is Adam with a learning rate of $5 \cdot 10^{-4}$. The change in the policy distribution is clipped at 0.2 using the PPO technique. The gradients are clipped to the norm by a value of 1, and a dropout with a probability of 0.1 is used in both the LSTM encoder and the Transformer encoder. A complete summary of the hyperparameters is available in Table 5.4.

Tab. 5.4.: Summary of the hyperparameters.

Hyperparameter	Value
<i>Learning rate</i>	0.0005
<i>Batch size</i>	800
<i>Instances per batch</i>	20
<i>No. of rep. per instance (N)</i>	40
<i>PPO Clipping factor</i>	0.2
<i>Entropy coefficient</i>	0.01
<i>Embedding size</i>	64
<i>Gradient clipped by norm</i>	1
.....	
<i>No. LSTM layers</i>	1
<i>LSTM hidden size</i>	16
<i>Dropout LSTM</i>	0.1
.....	
<i>No. Transformer layers</i>	2
<i>No. heads in the self-attention</i>	2
<i>Dropout Transformer</i>	0.1

5.3.2 Performance comparison

In order to perform a comparison, other classical alternatives are evaluated. In particular, due to the specific networking conditions assumed in this problem, it is hard to find heuristics that fit this exact problem statement. This fact has no impact on the RL approach, as it treats the problem as a black-box. After weighing the alternatives, the comparison is conducted using a Genetic Algorithm (GA) [8], as it gives a good representation on the performance of metaheuristics, and a constraint solver (CP). In regard with this last, the non-linearities on the problem lead to *if-else* statements that come natural to CP engines. Alternatively, these *if-else* statements can be relaxed and transformed into linear constraints, which would allow to also use MP for its resolution. Mayor MP solvers include conditional statements (see Annex A.2). Although, in the case the problem presents complex relation between constraints, generally CP represents a better option. Due to this reason, the IBM CPLEX-CP solver [55] is used on this problem.

With regard to the GA, the hyperparameter settings are a population of 600 individuals, a crossover rate of 0.8 and a mutation rate of 0.3. The selection mechanism is a tournament done over the best of 3. The algorithm runs 50 generations before stopping (enough iterations to converge in the different problems included in the study).

Tab. 5.5.: Average objective, standard deviation and mean computing time for instances of the VNF-CE problem. The recurrent model (a) is depicted as RL_S and the fully attentional model (b) as RL_T . The size of the instance is denoted by the number of hosts in the infrastructure and the length of the service chain m .

Method	m=5			m=7			m=9			m=11		
	mean	std	time	mean	std	time	mean	std	time	mean	std	time
(VNF-CE10)												
GA_P(600)	1.627	0.64	1.51s	3.118	1.88	1.72s	4.544	3.32	2.14s	4.959	2.42	2.38s
RL_S(1)	2.040	1.21	0.024s	3.751	2.40	0.030s	5.422	3.91	0.036s	6.045	3.07	0.038s
RL_S(40)	1.630	0.64	0.036s	3.158	1.86	0.042s	4.605	3.56	0.044s	4.932	2.45	0.050s
RL_S(100)	1.619	0.63	0.050s	3.139	1.84	0.063s	4.554	3.47	0.812s	4.834	2.38	0.096s
RL_T(1)	1.962	0.97	0.027s	3.828	2.26	0.032s	5.604	4.07	0.036s	6.473	3.31	0.038s
RL_T(40)	1.623	0.64	0.045s	3.142	1.86	0.044s	4.574	3.51	0.068s	4.943	2.48	0.077s
RL_T(100)	1.619	0.63	0.088s	3.146	1.85	0.12s	4.540	3.46	0.172s	4.872	2.40	0.206s
CPLEX-CP	1.615	0.62	0.264s	3.086	1.83	1.73s	4.418	3.27	98.1s	4.720	2.31	574.8s
(VNF-CE20)												
GA_P(600)	2.010	0.56	1.67s	2.857	0.58	1.84s	3.915	1.14	1.97s	4.545	0.93	2.25s
RL_S(1)	2.446	0.83	0.029s	3.243	0.93	0.030s	4.368	1.48	0.036s	4.874	1.25	0.038s
RL_S(40)	2.009	0.59	0.041s	2.798	0.59	0.053s	3.852	1.19	0.060s	4.306	0.89	0.065s
RL_S(100)	2.003	0.58	0.051s	2.786	0.58	0.078s	3.801	1.16	0.084s	4.300	0.89	0.092s
RL_T(1)	2.208	0.70	0.027s	2.980	0.68	0.030s	4.168	1.55	0.033s	5.136	1.57	0.038s
RL_T(40)	2.005	0.58	0.044s	2.798	0.62	0.061s	3.863	1.22	0.064s	4.386	1.00	0.062s
RL_T(100)	2.003	0.58	0.115s	2.776	0.57	0.156s	3.797	1.17	0.145s	4.313	0.90	0.124s
CPLEX-CP	1.986	0.57	3.92s	2.770	0.57	444.6s	3.745	1.11	1h(*)	4.263	0.86	1h(*)
(VNF-CE50)												
GA_P(600)	2.070	0.49	4.78s	2.936	0.57	2.63s	3.780	0.79	2.61s	4.750	0.78	2.80s
RL_S(1)	2.290	0.66	0.030s	3.024	0.82	0.032s	3.776	0.96	0.036s	4.650	0.82	0.037s
RL_S(40)	2.002	0.49	0.039s	2.799	0.58	0.050s	3.511	0.78	0.053s	4.380	0.68	0.061s
RL_S(100)	2.002	0.49	0.064s	2.797	0.58	0.075s	3.515	0.79	0.083s	4.370	0.67	0.094s
RL_T(1)	2.301	0.74	0.027s	3.185	0.99	0.030s	3.800	0.80	0.038s	4.777	1.01	0.039s
RL_T(40)	2.007	0.48	0.064s	2.810	0.57	0.056s	3.550	0.76	0.058s	4.365	0.69	0.067s
RL_T(100)	2.002	0.48	0.149s	2.803	0.58	0.114s	3.537	0.75	0.108s	4.369	0.68	0.121s
CPLEX-CP	1.998	0.49	383.2s	2.775	0.57	1h(*)	3.501	0.77	1h(*)	4.310	0.69	1h(*)

(*) The result is not optimal, the execution has been forced to end after the indicated time.

Finally, it is worth mentioning that towards seeking a fair comparison, the constraint solver and also the metaheuristic algorithm deal with the same problem statement as the RL model. This is, the relaxed variant where the occupancy, networking and latency constraints are introduced into the objective function using the Lagrange multipliers.

5.3.3 Results

The results on the VNF-CE problem are summarized in Table 5.5. Particularly, the average objective, standard deviation and mean computing time for the different instances of the problem is noted. Regarding the resolution methods, the recurrent model (a) is referenced as RL_S and the fully attentional model (b) as RL_T , followed by the number of the repetitions per instance N in brackets. E.g., $RL_S(40)$ indicates that the recurrent model (a) has been used with a self-competing breadth of $N = 40$.

This problem presents some features that benefit the learning process when compared to the previous JSP and this is reflected in the results. In this case, services are synchronously placed one at a time, the model selects the "n" element in the chain and places it. Also, the output is a categorical distribution over the hosts in the infrastructure (instead of binary scheduling decisions). These factors make it easier for the model to extract features from the problem definition and perform better decisions. This also benefits in a lower number of parameters of the neural network and faster learning times.

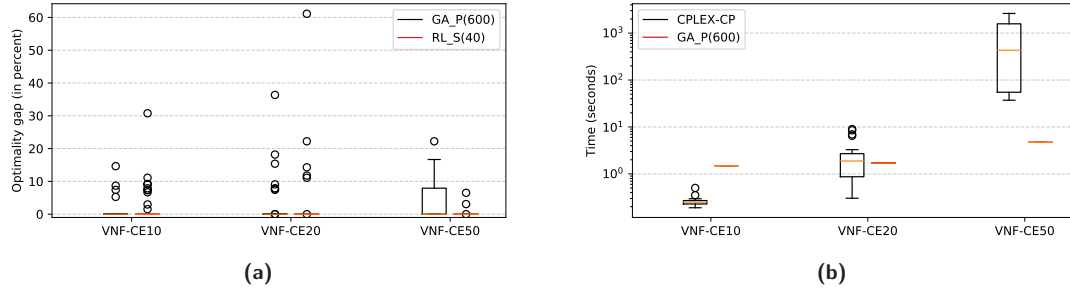


Fig. 5.6.: Optimality gap (a) and inference computational time (b) for the VNF-CE problem. The optimal solution is obtained for a problem with small service lengths $m = 5$, this is done using a constraint programming solver. The $RL_S(40)$ model is compared in performance and computational time against a genetic algorithm and the constrain solver itsef. The results show that the RL model obtains competitive solutions in shorter inference time.

From the obtained results we conclude that in this problem the NCO model achieves close to optimal solutions. In this scenario, and as previously discussed, the CP solver is competitive only when the size of the problem is not large and the complexity for achieving optimal placements is low. For larger instances of the problem, the CP solver does not represent a competitive alternative due to the high computational time it requires. In that case, the NCO model becomes a more suitable alternative. In this sense, it achieves even better results than the GA. Despite this problem represents a good scenario for metaheuristics, as getting feasible solutions is not particularly difficult, the NCO model gets better solutions in value and it achieves them in a fraction of the time. Particularly, we observe that **both the recurrent model (a) and the attentional model (b) achieve very similar results**. Hence, we deduce that both models represent a good alternative for embedding the required sequences. Finally, it can be also concluded that **a self-competing breadth of $N = 40$ provides a reasonable tradeoff between computation and performance**.

A more exhaust comparison on the small instance $m = 5$ of the VNF-CE problem is depicted in Fig. 5.6. In that case, the optimal solution can be obtained using the CPLEX-CP solver in reasonable times, so that an optimality gap comparison can be performed. As Fig. 5.6a shows, the RL model consistently predicts close to the optimal solutions. In this scenario, **the model is able to extract positive behaviours from the whole combinatorial space and infer a policy that almost suits perfectly on the problem instances**. Although, as argued in Chapter 4, the optima for all cases cannot be achieved using bare RL. On these small instances on the problem the NCO model gets similar performance than the GA. Although, as depicted in Fig. 5.6b, the times each alternative requires are quite different. The computational time on the GA, although it does not grow exponentially with the size of the problem as occurs with the CPLEX-CP solver, is sufficiently large to surpass the time threshold required on the solution. In this sense, only heuristics (handwritten or autonomously learnt on the problem) represent a viable alternative.

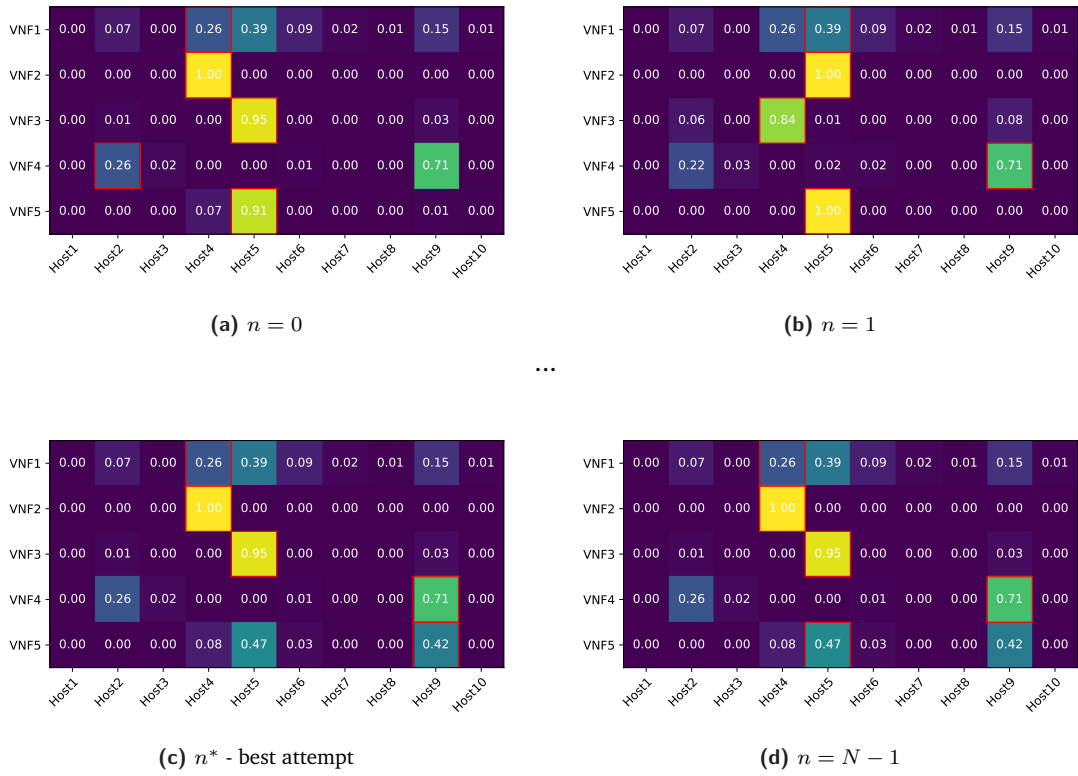


Fig. 5.7.: Action probability distribution of the $N = 40$ solutions inferred following the self-competing strategy on a testing VNF-CE10 problem example with a $m = 5$ service request. The particular action sampled from the distribution in each case is marked in a red box.

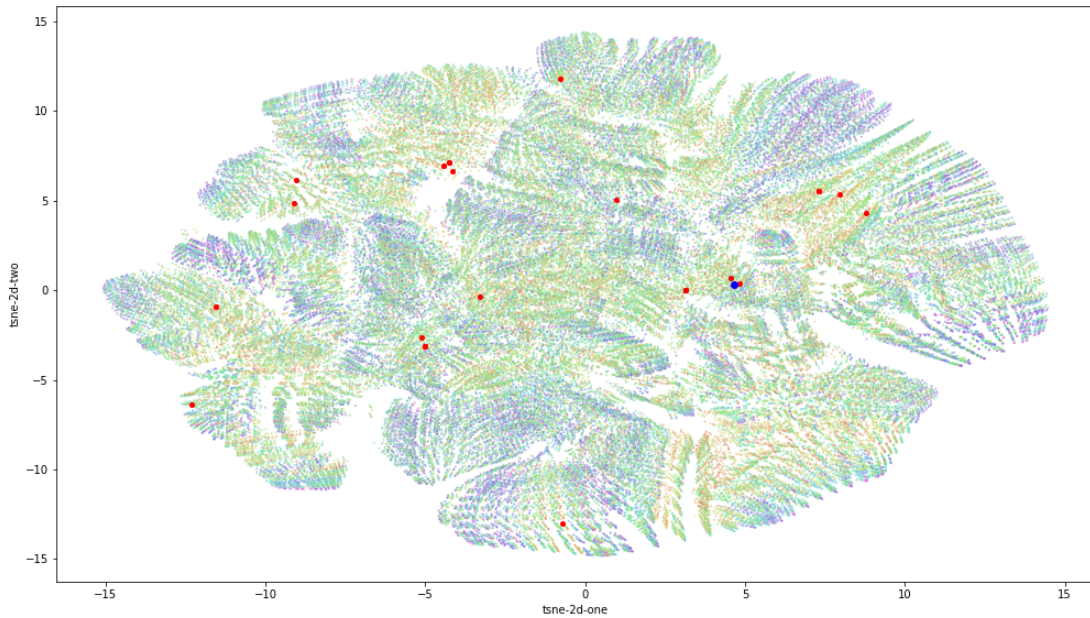


Fig. 5.8.: The t-SNE dimensionality reduction technique is used to plot the combinatorial space (Ω) on a VNF-CE10 problem example. The possible solutions on the problem are colored indicating the cost obtained on each placement. Warmer colors represent a smaller cost, while cooler colors a higher operational cost. The N solutions the model obtains are overlapped using red dots; the best attempt n^* is indicated using a blue dot.

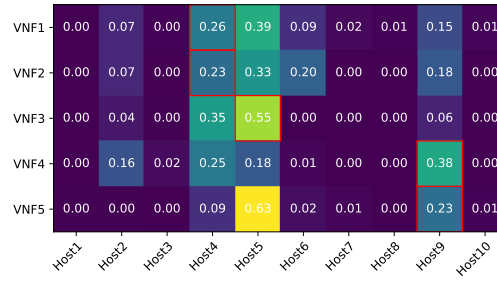


Fig. 5.9.: This figure depicts the average probability distribution computed on the $N = 40$ inferences obtained on the testing VNF-CE10 problem instance. This artificially created probability distribution gives an idea on the breadth experienced during the sampling. The best attempt n^* is marked using red boxes.

Regarding the self-competing strategy, this method computes N possible solutions on each problem instance in order to estimate the performance of the policy and build an unbiased baseline estimator. In Fig. 5.7 the $N = 40$ solutions the $RL_S(40)$ model obtains on a testing VNF-CE10 problem instance with a $m = 5$ service request is depicted. This heatmap plots the output probability distributions the policy observes at each state on the resolution process. For each of the N parallel episodes the model experiences upon the same instance, this picture tracks the action probability distribution the model gets over the action space (the hosts where to place each VNF). At every step on the resolution process the model defines its actions sampling over this probability distribution. The particular actions the agent takes are highlighted using a red box.

As observed, this learning strategy obtains an action probability distribution that it is not deterministic. Otherwise, the model would experience the same result on the N episodes and would not experience any benefit from this strategy. The model optimizes the breadth of the action distribution according to the number of samples performed. In this example, $N = 40$, hence the model produces a narrow the probability distribution over a few hosts from where the solutions would be sampled.

Also, the t-SNE dimensionality reduction technique can be used to picture the combinatorial space (Ω) on this problem example (see Fig. 5.8). To this end, all the possible solutions are computed on the problem. The t-SNE depicts close in space, combinations that do not differ much in the host selection. In addition, the solutions are colored. Warmer colors represent placements with low energy consumption whereas cooler colors represent placement decisions with high cost. As can be observed, this represents a high non-convex space where slight differences in the combination can produce a complete different cost. Over this combinatorial space, the N solutions the model computes are marked in red. In addition, the best attempt n^* obtained in the sampling is indicated using a blue mark.

Finally, in order to get an idea on the breadth of the search, we average the output distributions experienced during the sampling. This is depicted in Fig. 5.9. This artificially created probability distribution summarizes the host explored on the different sampled solutions.

5.3.4 Learning and inference times

As mentioned, the run times in the VNF-CE problem are considerably shorter than the presented in the JSP example. This is due two main factors: first, the size of the sequences, which determine

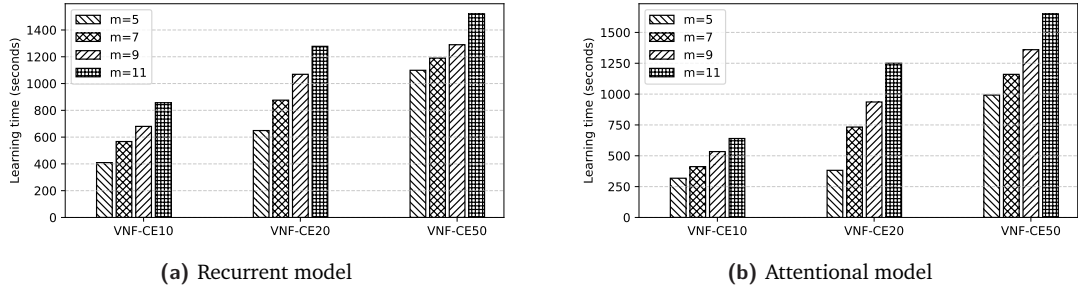


Fig. 5.10.: Learning time for (a) the recurrent model $RL_S(40)$ and (b) the attentional model $RL_T(40)$ on instances of the VNF-CE problem with service chains of different lengths m .

the number of iterations the model needs to perform to get the solutions, is much shorter in the VNF-CE; and secondly, the number of parameters used in the neural model is considerably lower. As a result, the computational time required to perform a single episode (compute a solution on the problem) is fast enough to achieve the real-time interaction pursued in this work (see Table 5.5).

In regard with the time required to learn the model, this is depicted in Fig. 5.10. As observed, the learning time is proportional to the number of operations, which grows with the number of hosts and the length of the services placed on the infrastructure. The learning intervals obtained are in the order of minutes for the scenarios studies. This meets the requirements advanced in Sec. 5.2. As mentioned, the network optimization cycles operate hourly/daily/weekly changing the network definition. In this scenario, the NCO model needs to be particularized for the network state, hence a new learn process is required at that frequency in order to adapt the model to the new situation. The learning times observed in both models are not excessive and made this method a suitable option for network optimization. Although we would like to stress that in this respect the recurrent model is preferred, as the learning time do not increase so severely with the complexity of the problem as the attentional model does.

5.4 Conclusions

In this Chapter a use-case for the NCO framework argued in this Thesis is presented. In particular, this is done within the scope of intelligent systems to orchestrate the 5G network. Here, the Virtual Network Function placement problem is addressed. The goal is to optimize at operational time the deployment of services within the infrastructure. To this end, the problem is mathematically formalized and subsequently solved.

In order to validate the NCO model a complete experimentation in this problem is conducted. The NCO model is benchmarked against metaheuristic algorithms and constraint programming solvers to disclose the best alternative on the problem. In this particular case, designing handwritten heuristics is not straightforward, thus heuristics are not introduced in the comparison. From the results obtained in the experimentation, **we conclude that both NCO models, the recursive and the attentional, represent a good alternative for the problem. Particularly, they outperform their counterparts when rapid solutions need to be obtained.** In this case, NCO outperforms the GA, used as a reference for metaheuristics, achieving better solutions in a fraction of the time. And the CPLEX-CP constrained programming solver does not represent a viable option when the

problem scales up in size. In addition, the inference time the NCO model achieves match the real-time requirements on the problem.

This Chapter also provides an insight on how the neural model produces solutions on the VNF-CE problem. A breakdown on the decoding process on this example is presented to get a better idea on the *self-competing strategy*. To this end, the output probability distribution from where the different solutions are sampled is analyzed. This gives a perspective on the breadth of the search the method produces on the combinatorial space (Ω).

After verification, we infer that **the time required to learn the model is suitable for the desired goal**. As mentioned, the network optimization cycles operate hourly/daily/weekly changing the network definition. In this scenario, the NCO model needs to perform a new learning process in order to adapt to the new network situation. The learning times observed in this experiment are not excessive and make this method a suitable option for network optimization.

Contents

6.1	Performance comparison between recurrent and attentional models for NCO	91
6.2	Graph neural networks applied to NCO	92
6.3	NCO in combination with Tree-Search strategies	92
6.4	RL to enhance Metaheuristic algorithms	93
6.5	Discussion on the ITU-T approach for introducing ML in 5G	93
6.6	Conclusions	94

Despite the NCO research line in which this Thesis is focused, other alternatives studied in the community are here presented. In this Chapter, different approaches to the problem are discussed, arguing their validity for the specific aim purposed, building real-time decision making systems to optimize network deployments.

Finally, the work of the *ITU-T study group focused in ML applied to 5G networks* is here exposed. An analysis of the different case of studies this entity is carrying on is done and a motivation for implementing the RL alternative developed in this work is suggested.

6.1 Performance comparison between recurrent and attentional models for NCO

In the literature the *Transformer network* [116] has become the new standard for building sequential NLP models. Also, this architecture has shown better performance when compared to recurrent sequence-to-sequence models for solving combinatorial problems [65, 35]. These referenced works decline the use of RNNs in favour of attention mechanisms. This is because RNNs are complex cells whose implementation usually degrade the representation of the individual elements, specially the distant ones. Until recently, recurrent cells were the best alternative for dealing with dynamic length sequences, both preserving the order of the elements in the chain or for building set-aware models that are invariant to the order [118]. Transformer networks instead use a positional encoding mechanism that allows to build an attentional model over them without requiring recursion.

In our experience, **we have achieved similar performance using both alternatives: recurrent and attentional models**. Although the setup used on the experimentation was different from the presented on the previous references, i.e. not the whole Transformer network is used. In our Markovian model, sequence embedding is only required in the encoder, where it is used to extract the features from the problem instance definition. This differs from the sequence-to-sequence models, where sequences have a bigger relevance on the model (both in encoder and decoder). During the validation on the VNF-CE problem we have observed that both recurrent and attentional encoders give a fair representation of the problem definition. Therefore, we conclude other parts of the model (e.g., using attention on the decoder) or a better hyper-parameter tuning have a bigger impact on the model performance. Differently, in the JSP the Transformer encoder

attends over a much larger representation, this increases significantly the memory this model requires. Hence, we dismiss this alternative.

6.2 Graph neural networks applied to NCO

Numerous combinatorial problems over different disciplines (social networks, transportation, telecommunications, etc.) are represented as graphs. In fact, of Karp's 21 problems [58], 10 of them are graph optimization problems while most of the other 11 problems also can be formulated as graphs. Graphs embedding throughout Graph Neural Networks (GNNs) has been pursued for long [96]; however, until recently this technology has not been applied to combinatorial problems. In particular, (khalil et al., 2017) [60] uses a graph embedding strategy [31] for exploiting the structure of these problems in order to learn an heuristic. As they argue, sequence architectures used e.g., for solving the TSP [120, 15] are generic structure that even though can be applied for addressing these problems, are generic and are not effective reflecting the structure of graph problems.

In our case, the natural representation for the VNF Placement problem is also a graph $G(V, E)$. Where the servers in the infrastructure are represented by nodes V and the edges E model the link interconnections. However, **graph representation using neural networks is a technology still in its early days**. For this reason, a simplification on the network interconnection has been established. This way, the VNF-CE can be formulated a sequence problem, and thus knowledge mainly developed in the NLP community can be applied to it.

6.3 NCO in combination with Tree-Search strategies

NCO has achieved competitive results in many different CO problems. However, these models heavily rely on the condition that positive rewards can be obtained from an initial random policy and enough exploration on the environment. This strategy is valid for numerous domains, although it struggles in some scenarios: e.g., in environments with sparse rewards as occurs in the famous Sokoban and Montezuma's revenge games; or for the interest of this Thesis, for achieving optimal decisions for the whole combinatorial space, as this is asymptotically difficult. To overcome these difficulties, RL has been successfully combined with search strategies to expand the exploration and improve the action selection.

In particular, excellent results have been obtained combining Policy Iteration (Annex B.2.2) with Monte Carlo Tree Search (MCTS) to learn *tabula rasa* through self-play [105]. In this approach, a neural network is used to provide a policy $\pi_{\theta}(\cdot|s)$ and a estimate state value function $\hat{v}_{\theta}(s)$ for every state s . This method performs a tree search focusing on moves the policy indicates to guide the search strategy. Namely, the policy is used to reduce the breadth and the value function is used to reduce the depth in the MCTS. Examples of this method are AlphaZero [105] and Expert Iteration [7].

We believe that harnessing search strategies will also lead to better RL approaches for combinatorial optimization. Support the model on a tree search helps to improve the exploration and to convergence the model to better behaviours. This strategy positions between the full backups explored in dynamic programming and traditional RL methods based on single backups (see Fig. B.3.1). Thus, this method works *smarter not harder* as reduces the width of bare tree searches and improves the results. E.g., the state-of-the-art Chess engine Stockfish evaluate up to

60 million moves, whereas AlphaZero achieves better results only evaluating around 60 thousand on the decision tree [104]. Other examples that use this technique can be found for solving the Rubik's Cube [75] or for dealing with Bin Packing problems [67]. In conclusion, this seems a promising strategy for approaching combinatorial problems. And although at inference results can take longer time to be obtained, the depth of the search can be tuned to be in accordance with the time available to achieve the solution.

6.4 RL to enhance Metaheuristic algorithms

A different approach explored in the literature for the purpose of this Thesis rather than inferring greedy heuristics, is in the usage of RL in combination with metaheuristic algorithms (previously introduced in Section 2.1.2). For example, [28] proposes to learn a RL model to support a local search procedure. An iterative process in which an existing solution is improved by rewriting local parts of the solution. Although, when metaheuristics are used to deal with highly constrained problems they present some concerns. In this case, paraphrasing their authors: "this formulation is especially suitable for problems with the following properties: a feasible solution is easy to find; and the search space has well-behaved local structure". In this case, a local search strategy struggles in constrained environments, as if the rewriting mechanism achieves an unfeasible solution, it discard the solution and repeats the process. This is a drawback that repeats on metaheuristic algorithms. Therefore, **this approach does not guarantee a rapid solution**. Unlike NCO, which learns a greedy heuristic that provides rapid approximations obtaining near feasible near optimal solutions even in highly constrained environments.

6.5 Discussion on the ITU-T approach for introducing ML in 5G

Currently, the ITU-T Focus Group on Machine Learning for Future Networks including 5G (ITU-FG-ML5G) is working in integrating an ML-aware architecture in the telecommunications network¹. In their resulting works, this group envisions an ML-based schema conformed by pipelines that are integrated in the 5G service-based architecture (SBA). Different nodes collect data from different origins, pre-process it, distribute them into ML-models, and redirects the output to the corresponding sinks in order to take the necessary decisions. These pipelines are distributed over the network, can be place on the user premises or on the core network. Finally, a ML Function Orchestrator (MLFO) aligned with the NFV MANO architecture (Sec. 5.1), manages the ML pipelines, being also responsible of selecting the ML models based on the needs of the ML applications.

One of the major concerns, is that although this architecture is not exclusive of supervised learning, all the practical cases discussed so far belong to this case [64, 94]. Supervised learning is used to train the models in different scenarios using optimal labels obtained from simulated environments. For later, deploy the different models prepared for different network conditions into this ML-aware architecture. In this schema, it is the MLFO's decision to selects the most suitable model in each scenario. These supervised models are focused on performing regression

¹The ITU-FG-ML5G work is aligned with the ITU-T standards for providing the architectural framework for the integration of machine learning into 5G (ITU-T Y.3172), the framework for evaluating intelligence levels across different parts of the network (ITU-T Y.3173), and the framework for data handling (ITU-T Y.3174).

and classification techniques. Henceforth, the ability they have to generalize from simulations to the real implementation is key for the success of the solution.

Alternatively, we propose using RL for this purpose. In this setup the model learns to adapt on a real scenario, discovering by itself how to act in it. This is a viable option as not optimal labels for each specific case are required for training the model. It is the model who interacts with the environment and discovers by itself how to act accordingly.

6.6 Conclusions

In this Chapter, different alternatives to NCO are discussed. Despite the Markovian NCO model, which aims at inferring end solutions on sequence CO problems, other alternatives can be used in that sense. Here, we argue different neural models for NCO but also expose other strategies that relying on RL can also be applied to address these problems. It is the case for example of metaheuristics supported by RL or in our vision a very promising strategy that mixes Policy Iteration with MCTS to perform a model-based learning. Particularly, this last strategy has had excellent results in strategic thinking and thus we believe that could benefit CO problems. This is because this technique relies on a decision tree, which enables to perform simulations on the future actions. This is a great step forward when compared to classical RL strategies in which decisions are taken without this prediction analysis.

Finally, the work of the ITU-T study group FG-ML5G is also analyzed. After studying the different case of studies these entities are carrying on, one observes that the vast majority of their proposals are models based on supervised learning. The different examples in which the study group is focused on (e.g., link adaptation optimization, channel prediction,...) are oriented on regression and classification. Models that are pre-trained from simulated network conditions, and whose ability to generalize is key for achieving a good performance in a real scenario. Here, we suggest to use models that based on the current state of the network an RL-based learning could be done to discover at each time a good heuristic to rely on for optimizing the network.

Final conclusions, contributions and broader impacts

Contents

7.1	Thesis coverage	95
7.2	Main contributions	96
7.3	Final conclusions	97
7.4	Thesis publications	97
7.5	Future Work	99
7.6	Broader Impacts	99

Finally, a summary of the most important aspects covered in this Thesis together with the final conclusions are here presented. This Chapter ends emphasizing the broader impacts of NCO and the repercussion of this work on the OR industry.

7.1 Thesis coverage

This Thesis gives a step forward in the use of RL for addressing combinatorial problems. It contributes directly to the NCO theory, but also to the research community in the form of conclusions that lead to a better understanding on the technology and results useful for further comparisons. In the following, the coverage of this Thesis is summarized.

- **This Thesis covers the use of NCO for solving CO problems that can be expressed as sequences.** To this end, different neural architectures and RL strategies applied to sequence models are evaluated. These include Recurrent Neural Networks (RNN) and fully-attentional models as the novel Transformer network. In order to implement these neural models, tensor oriented libraries are used, specifically Tensorflow and PyTorch. Also, a study on the different learning strategies that can be applied to CO is conducted on this work. This includes policy-based and model-based learning strategies.
- **This work also covers the constraint management within the NCO framework.** Existing works do not deal with constraints, they build specific models to ensure feasible solutions or rely on masking schemes to avoid unfeasible actions. We argue that actions that lead to an immediate violation of a constraint should be masked to directly avoid exploring those infeasible solutions. However, due to their nature, not all constraints can be verified before acting, and thus, cannot be masked. This work covers this gap in the literature and proposes to apply *Reward constrained policy optimization* technique into NCO to soften these constraints and incorporate them into the objective function.
- **A methodology for addressing CO problems is also defined.** To this end, a toy reference combinatorial problem is used to explore the learning capabilities of the model. This study inquires in the different challenges that come along when solving CO problems using RL and paves the way towards optimizing the learning process. In this sense, different methods

to enhance the results are tested. Specifically, this work emphasizes the importance of trust-region optimization to avoid performance degradation and entropy regularization to extend the learning process and gain a better convergence on the process.

- Also, due to the importance observed in the selection of the hyperparameter for the outcome of the model, we conclude that a **full-vectorization** of the problem is required to reduce the learning times and therefore better fine-tuning the model. We contribute in this direction designing a self-competing algorithm (Algorithm 1) that benefits from this tensor-based implementation in two aspects: first, reducing the complexity of the model, as it does not require a critic estimator to predict the performance of the policy network, this information is extracted from the current policy distribution; and second, enhancing the exploration and selection of the solution as it operates as a breadth search strategy.
- Finally, **the proposed approach is validated in the Job Scheduling Problem (JSP)**, a reference CO problem where a complete set of heuristics are available for comparison. And ultimately, in the **VNF chain embedding problem (VNF-CE)**, a problem present in today's telecommunication infrastructure. Conducted experiments prove that there exists an optimal range where the proposed solution is superior for computing rapid solutions when compared to classical heuristics, metaheuristics, and Constraint Programming (CP) solvers. Particularly, in the low to medium range of the problem sizes, the NCO model infers a heuristic behaviour that outperforms classical handwritten heuristics. However for larger size problems this advantage fades and the policy performs similar to the classical algorithms.

7.2 Main contributions

The contributions here presented fall into two main categories. Firstly, this work argues that the performance obtained by sequence-to-sequence models used for NCO in the literature is improved raising combinatorial problems as Constrained Markov Decision Processes (CMDP), such property can be exploited for building a Markovian model that computes the solutions incrementally based on interactions with the problem. This benefits the model as relying on the immediate states during the resolution gives to the model a better understanding on how the solution evolves in the problem, and therefore, improves the quality of the results. It is better for the model to act based on the actual representation of the problem rather than doing on memories as previous models did. Secondly, this work presents a strategy to extend NCO to constrained combinatorial problems. Specifically, it is argued that masking schemes shall be used to deal with the constraints if the problem formulation allows its implementation. Namely, it is beneficial for the model to mask actions that lead to an immediate violation of a constraint to directly avoid exploring those infeasible solutions. However, due to the nature of combinatorial problems, not all constraints can be verified before acting, and thus, cannot be masked. To that end, this work proposes dealing with non-maskable constraints by incorporating them into the optimization objective. Conducted experiments prove that this contribution allows to efficiently apply NCO to general constrained combinatorial problems.

This work also contributes providing a methodology for addressing CO problems. In order to achieve an optimal learning strategy, a procedure for building the model is proposed. To this end, it is crucial to estimate the learning capabilities of the models, detect any possible flaw on the learning process and use enhance learning methods to address these issues. This methodology

serves as a reference strategy to apply when addressing any other CO problem and gives a hint on the possible drawbacks that may appear and provides an intuition on how to solve them.

7.3 Final conclusions

Combinatorial optimization is being solved by thousands of companies around the world every single day. In some instances, they can wait for hours to obtain sufficiently good solutions, but more and more industrial players rely on the ability to get fast (sometimes in real-time) high accurate solutions.

Through the experimentation presented in this work, it is proved that **NCO is capable of computing near-feasible near-optimal solutions with rapid inference times, outperforming traditional OR approaches under the presented circumstances.** The inference time obtained in NCO models is only comparable to greedy heuristics. As metaheuristics and CP solvers require much larger times to obtain a solution, and thus, are non-compatible with the real-time decision making system seek in this work. Particularly, the model here presented demonstrates that for small to medium size CO problems, the policies achieved are competitive when compared to the alternatives. In this sense, **obtaining better solutions quicker without human intervention can significantly reduce costs and make industry players more competitive.**

This approach is validated performing a complete experimentation in a classical combinatorial problems, the JSP and constrained variants of it. But also in a real scenario, in this sense **NCO is used for building a real-time decision-making system that optimizes the placement of virtual network functions on a 5G deployment.** This use-case proves that the proposed architecture presents the required versatility for being applied in real-world problems. This experiment evidence that NCO is competitive in the results, that the inference times obtained are within the expected limits and that the leaning process matches the requirements for being used in network optimization.

7.4 Thesis publications

This Thesis has result in numerous publications. This section resumes the journal articles as well as conferences and congresses this work has been presented.

Publications

- Solozabal, R., Ceberio, J., Sanchoyerto, A., Zabala, L., Blanco, B., & Liberal, F. (2019). Virtual Network Function Placement Optimization With Deep Reinforcement Learning. *IEEE Journal on Selected Areas in Communications*, 38(2), 292-303.
- Solozabal, R., Blanco, B., Fajardo, J. O., Taboada, I., Liberal, F., Jimeno, E., & Lloreda, J. G. (2017, August). Design of virtual infrastructure manager with novel VNF placement features for edge clouds in 5G. In *International Conference on Engineering Applications of Neural Networks* (pp. 669-679). Springer.
- Solozabal, R., Sanchoyerto, A., Atxutegi, E., Blanco, B., Fajardo, J. O., & Liberal, F. (2018). Exploitation of mobile edge computing in 5G distributed mission-critical push-to-talk service deployment. *IEEE Access*, 6, 37665-37675.

- Solozabal, R. et al., (2018, May). Providing mission-critical services over 5G Radio Access Network. In *IFIP International Conference on Artificial Intelligence Applications and Innovations* (pp. 520-530). Springer.
- Solozabal, R., Fajardo, J. O., Blanco, B., & Liberal, F. (2018). Quality of Service (QoS) oriented management system in 5G cloud enabled RAN. *XIII Jornadas de Ingeniería telemática (JITEL 2017)*. Libro de actas, 170-175.
- Kourtis, M. A. et al., (2019). A cloud-enabled small cell architecture in 5G networks for broadcast/multicast services. *IEEE Transactions on Broadcasting*, 65(2), 414-424.
- Sanchoyerto, A., Solozabal, R., Blanco, B., & Liberal, F. (2019). Analysis of the Impact of the Evolution Toward 5G Architectures on Mission Critical Push-to-Talk Services. *IEEE Access*, 7, 115052-115061.
- Carreras, A. et al., (2019). Impact of front-haul delays in non-ideal cloud radio access networks. *Wireless Personal Communications*, 106(4), 2005-2022.
- Zabala, L., Solozabal, R., Ferro, A., & Blanco, B. (2018, November). Model of a Virtual Firewall Based on Stochastic Petri Nets. In *2018 IEEE 17th International Symposium on Network Computing and Applications (NCA)* (pp. 1-4). IEEE.
- Spada, M. R., Pérez-Romero, J., Sanchoyerto, A., Solozabal, R., Kourtis, M. A., & Riccobene, V. (2019, June). Management of mission critical public safety applications: the 5G ESSENCE Project. In *2019 European Conference on Networks and Communications (EuCNC)* (pp. 155-160). IEEE.
- Blanco, B. et al., (2020, June). Intelligent Orchestration of End-to-End Network Slices for the Allocation of Mission Critical Services over NFV Architectures. In *IFIP International Conference on Artificial Intelligence Applications and Innovations* (pp. 74-83). Springer.
- Pérez-Romero, J. et al., (2019, December). Supporting Mission Critical Services through Radio Access Network Slicing. In *2019 International Conference on Information and Communication Technologies for Disaster Management (ICT-DM)* (pp. 1-8). IEEE.
- Sanchoyerto, A. et al., (2019, May). Orchestration of Mission-Critical Services over an NFV Architecture. In *IFIP International Conference on Artificial Intelligence Applications and Innovations* (pp. 70-77). Springer.
- Zabala, L., Ferro, A., Solozabal, R., & Blanco, B. (2018, October). Performance Analysis of a Network Sensor's Packet Processing System using Generalized Stochastic Petri Nets. In *Proceedings of the 15th ACM International Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, & Ubiquitous Networks* (pp. 63-70).

Conferences and congresses

The work here presented has been exposed in the following conferences and congresses:

- Modelling and Optimization: Theory and Application (**MOPTA**)
Combinatorial optimization with Deep Reinforcement Learning.
Lehigh University, PA (USA), 2019.
- Artificial Intelligence Applications and Innovations (**AIAI**)
Providing mission-critical services over 5G Radio Access Network.
Rodhes (Greece), 2018.
- Engineering Applications of Neural Networks (**EANN**)
Design of Virtual Infrastructure Manager with novel VNF placement features for edge clouds in 5G.
Athens (Greece), 2017.

7.5 Future Work

Many different model variants, tests, and experiments have been left to future researches. This Section concerns a deeper analysis of particular mechanisms and proposals to try on this methods. These can be summarized in the following ideas:

- This Thesis has been mainly focused on the use of NCO for solving CO problems that can be defined as sequences, and most of the neural architects used in the experimentation were obtained from the NLP literature or adapted from it, leaving the study of other architectures outside the scope of the Thesis. Although, several of the well-known CO problems in the literature can be addressed using this sequential approach (e.g., TSP, VRP, JSP), this representation is not the most suitable one for addressing problems with more complex relations. From our perspective, exploring more natural representations could provide significant benefits to NCO. In this sense, GNNs could be used to extend this theory to graph-oriented problems (see Sec. 6.2). This representation treats graph problems in their natural structure, extending the problems that can be addressed using this technology. Although GNNs are a promising technique for addressing problem representations otherwise impossible, these complex structures require further research to be competitive.
- Improvements on the results can be also obtained by researching additional learning strategies. New approaches in this direction can be induced from techniques such as the hybridization of RL with tree-search strategies (previously discussed in Sec. 6.3). NCO could benefit from this approach to perform a better exploration that could lead to better solutions. Also, following this strategy the action selection is improved as tree-search strategies allow to focus on the best option explored during the inference on the tree. However, implementing this learning strategy implies fundamental changes over the methods addressed in this Thesis. In that sense, implementations as OpenGo [114] (open source code inspired by DeepMind's AlphaGoZero) can be used to facilitate that process.
- Lastly, recent advances in exploration techniques have proved that RL can be used to address environments difficult to explore. Recent works such as Random Network Distillation [26], Never Give Up [10] or lastly Agent57 [9] have achieved remarkable results in Atari games where previous approaches were unable to operate. Specifically, Agent57 has been the first general AI capable of outperforming human in the whole Atari benchmark. In this sense and although this curiosity-driven strategy has also been tested in this work with negative results, further studies can be carried on to extrapolate these advanced exploration techniques to CO problems.

7.6 Broader Impacts

We want to stress that this is the first step in NCO towards much more complicated problem settings (i.e., stochastic problems or partially observable problems, to name a few) that both academicians and, more importantly, industrial partners would seek to solve using this technology. Addressing these problems is very complicated using traditional OR techniques. While for NCO, these problems come natural to the method. From the NCO perspective addressing these problems is straightforward, the model is able to learn patterns either in static environments (as demonstrated in this work) or stochastic problems where there exists an input distribution and the model needs to react based on a probabilistic approach. In both cases, the model seeks to learn a behaviour that

statistically fits well on the problem, which is unknown for the agent and treated as a black-box. Due to this reason NCO is envisioned as a breakthrough technology that would allow to address CO problems today intractable.

This Appendix A discusses further details on the conceptual differences between the exact resolution methods in OR described in this work: Mathematical Programming and Constraint Programming.

A.1 Mathematical Programming (MP) vs Constraint Programming (CP)

The model-and-run of MP optimizers

On one side, Mathematical Programming relies on applied mathematics to relax the problem and guide the tree search (e.g., "Branch and Bound"). In short, these are the concepts on which MP works:

- **Relaxation.** MP optimizers require to classify the problem into a well-defined mathematical category e.g., Linear Programming (LP), Mixed Integer Quadratic Programming (MIQP). This is due to the fact that MP, in the context of discrete optimization, uses relaxations techniques and cutting-planes strategies that require a mathematical knowledge on the problem.
- **Lower bound and optimality gap.** A mathematical programming engine will use different techniques such as a lower bound proof provided by cuts and linear relaxation to estimate the optimality gap.

The model-and-run of CP optimizers

On the other hand, Constraint Programming aims to solve CO from a different perspective. A CP engine does not make any assumption on the mathematical properties of the problem. CP uses logical inferences to optimize the search procedure. This allows the method to address difficult allocation, sequencing and scheduling problems otherwise intractable.

These are the concepts on which CP generates workable solutions on CO problems arising from such complexity:

- **Aggressive elimination on the search tree.** Domain reduction is extensively used to reduce the search on the decisions. Every reduction makes the problem easier since the optimizer propagates the results of the decision throughout the model.
- **Rapidly traversing the decision tree.** The concept of systematically exploring a decision tree for efficient solutions relates to domain reduction. This approach gives to the model the ability to move flexibly throughout the search space and rapidly to backtrack when early choices turn out to be dead ends.
- **Adaptive Search.** The default search consists of several search techniques that are dynamically changed during the search to adapt the problem. The different search techniques include but are not restricted to Large Neighborhood Search and Genetic Algorithms.

- **Discrete.** It should be noted that CP supports only discrete decision variables (integer or boolean), while a mathematical programming model supports either discrete or continuous decision variables.
- **Proof of optimality.** Lastly, note that a constraint programming engine proves optimality by showing that no better solution than the current one can be found.

A.2 Logical Conditions

Logical conditions, and specifically *if-else* statements, under certain circumstances can be formulated into a mathematical program using big-M methods. To achieve this, conditionals are transformed into dual constraint equations that compare the result against a big enough constant. These transformations rely on highly constrained equations using big numbers to ensure its satisfiability. However, this strategy presents some drawbacks. Big constant on the resolution may cause sparse resolution matrices, which in the end delay the resolution process. Due to this reasons, many modern MP solvers have indicator constraints, which facilitates this process as one can write implications directly without big-M constraints ¹, instead the solver choose an appropriate one.

Despite that, in the case the problem presents numerous or complex logical conditions, generally using constraint programming is a more recommended approach.

¹The IBM CPLEX-MP solver includes the *if-then* statement as part of its API.

” *Just as electricity transformed almost everything 100 years ago, today I actually have a hard time thinking of an industry that I do not think AI will transform in the next several years.*”

— **Andrew Ng**
Co-founder of Coursera

This Appendix B, provides a mathematical introduction on Reinforcement Learning. Particularly, (1) defines the Bellman equations for Markov Decision Processes, in (2) the Bellman equations are solved using Dynamic Programming; this leads to model-free learning where (3) Value-based learning, (4) Policy-based learning and finally, (5) Actor-Critic methods are covered.

B.1 Markov Decision Process

B.1.1 Definition

The mathematical basis for Reinforcement Learning (RL) is the Markov Decision Process (MDP) formulation. An MDP describes the iteration with a controllable dynamical system. It formalizes a decision process with an environment in which by definition all states satisfy the Markov property. An MDP is defined by a tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$, formed by the representational spaces and the one-step dynamics of the environment. Formally:

- \mathcal{S} is the state space.
- \mathcal{A} is the action space.
- \mathcal{P} is the state transition probability matrix $\mathcal{P}(s'|s, a) : \mathcal{S} \times \mathcal{A} \times \mathcal{S} \rightarrow [0, 1]$, which defines the probability of reaching state $s' \in \mathcal{S}$ from being in state $s \in \mathcal{S}$ and applying the action $a \in \mathcal{A}$.

$$\mathcal{P}(s'|s, a) = \mathbb{P}(S_{t+1} = s' | S_t = s, A_t = a)$$

- \mathcal{R} is the reward function, $\mathcal{R}(s, a) : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$, denotes the immediate reward for being in state $s \in \mathcal{S}$ and doing action $a \in \mathcal{A}$.

$$R_{t+1} = \mathcal{R}(S_t = s, A_t = a)$$

Given a controllable MDP, the dynamics of the decision process operates in the following way. Being in an arbitrary state $s \in \mathcal{S}$ at time step t , the agent chooses an action from the action space $a \in \mathcal{A}(s)$. The actions available at the moment t may depend on the current state S_t , or in other words, the actions available may be a subset of the whole action space $\mathcal{A}(S_t) \subseteq \mathcal{A}$. Due to the interaction with the environment, the system evolves to the next state $s' \in \mathcal{S}$ according to the transition function $\mathcal{P}(s'|s, a)$. As a result, the environment returns a one-step reward R_{t+1} . This process is repeated until a terminal state is reached. The generated sequence of states and actions created until the episode is ended is called the *trajectory*.

As mentioned, in an MDP all states present the Markov property. This property states that the state transition on the system only depends on the current system state S_t and the action A_t selected. Consequently, the transition dynamics are independent of the past trajectory, i.e. the previous states and actions. The current state captures all the information from the history.

Definition B.1.1 *Markov property:*

A stochastic decision process with states $S_t \in \mathcal{S}$, actions $A_t \in \mathcal{A}$ and a transition function \mathcal{P} is called Markovian if for every $t \in \mathbb{N}$ satisfies that

$$\mathbb{P}(S_{t+1}|A_t, S_t) = \mathbb{P}(S_{t+1}|A_t, S_t, A_{t-1}, S_{t-1}, \dots, A_0, S_0)$$

MDPs are classified according to several criteria. Depending on whether the state space \mathcal{S} and action spaces \mathcal{A} are discrete or continuous, the MDP is classified as discrete or continuous. With regard to the transition function $\mathcal{P}(s'|s, a)$, an MDP is classified as deterministic if there is no randomness in the transition process, and stochastic otherwise. Also, the reward function can be stochastic, in that case, the reward does not only depend on the state S_t but also in the stochasticity of the reward process itself. Due to this stochasticity in the reward, defining the mean expected reward is useful to further down formalize the equations that define the Markov process.

$$\bar{\mathcal{R}}(s, a) \doteq \mathbb{E}[R_{t+1}|S_t = s, A_t = a]$$

However, the particular immediate reward is not of major interest in the process, as an action that produces high immediate reward might lead to lower rewards in the long run. In an MDP the goal is to maximize the cumulative reward obtained during the episode. This return or accumulated reward over time is usually defined with a discount factor $\gamma \in [0, 1]$, which avoids to produce infinity loop returns in processes are able to run indefinitely. The cumulative return obtained at every time step in the trajectory is defined as

$$G_t \doteq R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-t} R_T = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \quad \forall t > 0$$

In this scenario, the ultimate objective is to find the policy π that leads to the maximum return G_t . The policy defines the behaviour of the agent, it represents the decision rules that determine how the agent acts in the environment. In an MDP, due to the Markov property, the agent only requires of the current state to act accordingly. Therefore, the policy can be established as a relation between the state and the action spaces $\pi : \mathcal{S} \rightarrow \mathcal{A}$. This rule for choosing an action by given the current state S_t , in general case is stochastic and can be viewed as a distribution over the actions $\mathcal{A}(S_t)$.

Definition B.1.2 *In a Markov Decision Process a stochastic policy π is defined as a probability distribution over the actions given the states,*

$$\pi(a|s) \doteq \mathbb{P}(A_t = a|S_t = s)$$

B.1.2 Value Functions

A common method for solving an MDP is through the definition of the value functions. The value functions estimate the value (expected return) of being in a given state and follow a predefined policy π . There are two different types of value functions: state-value function that only consider the state s , and action-value functions that indicate the quality of taking an action a on a given state s . They are formally defined as:

Definition B.1.3 *The state-value function $v_\pi(s) : \mathcal{S} \rightarrow \mathbb{R}$ of an MDP is defined as the expected return starting from state $s \in \mathcal{S}$ and following the policy π :*

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t | S_t = s]$$

Definition B.1.4 *The action-value function $q_\pi(s, a) : \mathcal{S} \times \mathcal{A} \rightarrow \mathbb{R}$ of an MDP is defined as the expected return starting from state $s \in \mathcal{S}$ and taking action $a \in \mathcal{A}(s)$ following the policy π :*

$$q_\pi(s, a) \doteq \mathbb{E}_{a \sim \pi}[G_t | S_t = s, A_t = a]$$

B.1.3 Bellman equations for the Value Functions

A convenient way to address the resolution of the value functions is using Dynamic Programming (DP). The term DP refers to a group of algorithms, that exploit the sequential property of multi-state decision processes to address their resolution. Particularly, Bellman proved that a dynamic optimization problem in discrete time can be stated in a recursive manner by writing down the relationship between the value function in one period and the value function in the next period. The relationship between these functions at consecutive time-steps is called the "Bellman equation".

Bellman expectation equations

To obtain the Bellman equations for the expected value functions, it is required to decompose the dynamics of the model into the relation of consecutive states. In the case of the state-value function $v_\pi(s)$, it can be expressed as the expectation of the immediate reward obtained in an interaction plus the discounted value function of the next state. Formally,

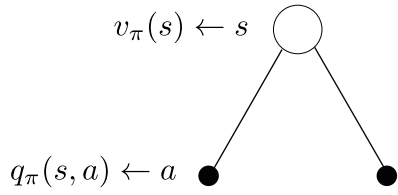
$$\begin{aligned} v_\pi(s) &= \mathbb{E}_\pi[G_t | S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots | S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma(R_{t+2} + \gamma R_{t+3} + \dots) | S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma G_{t+1} | S_t = s] \\ &= \mathbb{E}_\pi[R_{t+1} + \gamma v_\pi(S_{t+1}) | S_t = s] \end{aligned} \tag{B.1}$$

The action-value function $q_\pi(s, a)$ is similarly decomposed,

$$q_\pi(s, a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1}) | S_t = s, A_t = a] \tag{B.2}$$

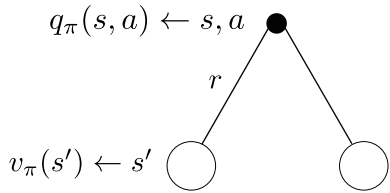
Following this procedure, the Bellman decomposition is extended to the whole one time-step dynamics of the problem. To that end, first the relation between state and action value functions is defined to further obtain the Bellman equations in their recursive form. The Bellman expectation

equation for the state-value function v_π obtained from the action-value function q_π is defined as



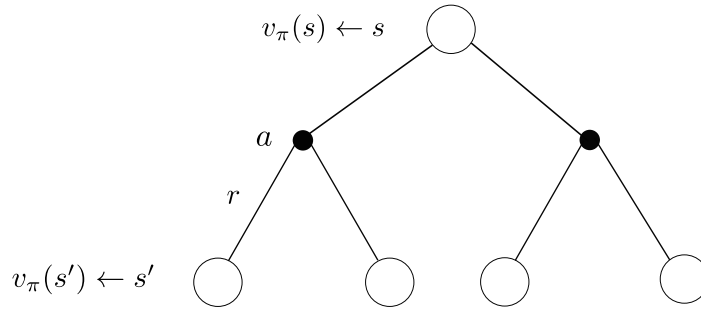
$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) q_\pi(s, a) \quad (\text{B.3})$$

Similarly, the Bellman expectation equation for q_π defined from the state-value function v_π is expressed as follows,



$$q_\pi(s, a) = \bar{\mathcal{R}}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, a) \cdot v_\pi(s') \quad (\text{B.4})$$

Finally, the Bellman expectation equations are derived by recalling the interconnection between the state (B.3) and action-value functions (B.4) in a recursive manner,



$$v_\pi(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\bar{\mathcal{R}}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, a) \cdot v_\pi(s') \right) \quad (\text{B.5})$$

$$q_\pi(s, a) = \bar{\mathcal{R}}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, a) \sum_{a' \in \mathcal{A}} \pi(a'|s') \cdot q_\pi(s', a') \quad (\text{B.6})$$

These equations define a linear system that can be analytically solved. Nevertheless, this is not practical for large MDPs, as the complexity increases exponentially with the number of states. In those cases, DP shall be used to solve the Bellman expectation equation.

Optimal Value Functions

It is worth noting that the value functions take different values according to the different policies. In this end, the optimal value function is the one that yields the maximum value compared to all other value functions. Mathematically the optimal value functions are expressed in the following form:

Definition B.1.5 The optimal state-value function $v^*(s)$ is defined as the maximum state-value function over all policies:

$$v^*(s) = \max_{\pi} v_{\pi}(s) \quad \forall s \in \mathcal{S} \quad (\text{B.7})$$

Definition B.1.6 The optimal action-value function $q^*(s, a)$ is defined as the maximum action-value function over all policies:

$$q^*(s, a) = \max_{\pi} q_{\pi}(s, a) \quad \forall s \in \mathcal{S}, a \in \mathcal{A} \quad (\text{B.8})$$

Optimal Policy

Achieving the best policy is the ultimate goal in solving an MDP. A policy π is defined to be better than or equal to a policy π' if its expected return is greater than or equal to that of π' for all states. In other words,

$$\pi \geq \pi' \iff v_{\pi}(s) \geq v_{\pi'}(s) \quad \forall s \in \mathcal{S}$$

For all MDPs there is always at least one policy that yields to the maximum reward when compared to all other policies, the optimal policy, denoted as π^* . The optimal policy π^* can be directly obtained from the optimal action-value function $q^*(s, a)$. Therefore, the optimal action-value function $q^*(s, a)$ embeds the optimal behaviour on the MDP, and obtaining it is equivalent to solving the MDP. Once the optimal action-value function is computed, obtaining the optimal policy π^* is direct, it is achieved maximizing over $q^*(s, a)$:

$$\pi^*(a|s) = \begin{cases} 1 & \text{if } a = \arg \max_{a \in \mathcal{A}} q^*(s, a) \\ 0 & \text{otherwise} \end{cases}$$

The Bellman optimality equations

Finally, the Bellman equations are also obtained for the optimal value functions, these are the *Bellman optimality equations*. The formulation of these equations is derive from the relation between the optimal value functions, that it is expressed as

$$v^*(s) = \max_a q^*(s, a) \quad (\text{B.9})$$

The Bellman optimality equations are non-linear equations that have no closed form solution, thus these equations are usually addressed using DP. To this end, the optimization problem is divided into a sequence of simpler subproblems, as Bellman's "principle of optimality" prescribes

Theorem B.1.1 *Principle of Optimality: Let assume an optimal action sequence $\{A_0^*, A_1^*, A_2^*, \dots\}$ resulting from an optimal policy π^* . Consider the sub-problem whereby one is at S_t at time t and wishes to maximize the return from time t . Then the truncated action sequence $\{A_t^*, A_{t+1}^*, A_{t+2}^*, \dots\}$ is also optimal for the subproblem.*

The Principle of optimality can be resumed in the fact that every optimal policy consists only of optimal sub-policies. This allows to break this decision problem into a sequence of subproblems that form the basis for DP. An MDP satisfy this principle (B.1.1), and it is the Bellman equation that indicates how to break down the optimal value function into the following steps: the optimal behaviour on the next time-step, followed by the optimal behaviour in the remaining trajectory.

Recalling equation (B.9), the Bellman optimality equations is obtained substituting from the optimal value functions,

$$v^*(s) = \max_a \left(\bar{\mathcal{R}}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, a) \cdot v^*(s') \right) \quad (\text{B.10})$$

$$q^*(s, a) = \bar{\mathcal{R}}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, a) \max_{a'} q^*(s', a') \quad (\text{B.11})$$

B.2 Planning by Dynamic Programming

This section details model-base planning on MDPs. Planning assumes the full knowledge of the model, namely the tuple $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$ that determines the dynamics on the problem. In this sense, MDPs can be solved for:

- **Prediction:** given an MDP and a policy π , output the value function v_π related to the model. For model prediction, the Policy Evaluation method is presented.
- Or **optimal control:** given an MDP model, obtain the optimal value function v^* , and therefore the optimal policy π^* . For optimal control Policy Iteration and Value Iteration methods are introduced.

B.2.1 Policy Evaluation

Policy evaluation is a model prediction technique. Based on an MDP model and a policy π , it judges how good is the given policy π . In other words, it computes the reward that it is expected to be obtained in the model following the given policy behaviour. Policy evaluation uses the Bellman expectation equation to calculate the state-value function v_π . For this purpose, it turns the Bellman expectation equation into an iterative update,

$$v_0 \rightarrow v_1 \rightarrow \dots \rightarrow v_k \rightarrow \dots \rightarrow v_\pi$$

$$v_{k+1}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \left(\bar{\mathcal{R}}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, a) \cdot v_k(s') \right) \quad (\text{B.12})$$

Policy evaluation operates as follows, it starts with an arbitrary function v_0 (e.g., all zeros) and proceeds updating it using DP. At each iteration k the value function is updated $v_{k+1}(s)$ based on the successor states $v_k(s')$. There are several techniques to proceed in the updates, the simplest one uses synchronous backups, which consider all states $s \in \mathcal{S}$ at every evaluation. This technique is proven to converge to the true value function v_π .

B.2.2 Policy Iteration

Related with optimal control, Policy iteration is the simplest technique to achieve the optimal behaviour. This method obtains the optimal policy for an the MDP though iterative improvements on the policy. This process consists in the following steps:

- **Policy evaluation:** in which the Policy evaluation technique is used to obtain the value functions for the current policy π .

- **Policy improvement:** in which the policy is improved by acting greedily on the resulted value function. Formally, the policy update is expressed as follows,

$$\pi'(s) = \arg \max_{a \in \mathcal{A}} q_{\pi}(s, a) \quad (\text{B.13})$$

This process is repeated iteratively to improve the policy until the Bellman optimality equation is satisfied, then the optimal value function is obtained $v_{\pi}(s) = v_*(s) \quad \forall s \in \mathcal{S}$. It can be demonstrated that Policy Iteration method converges to optimal policy π^* after enough iterations, although this process can be slow. To accelerate the convergence, instead of evaluating the complete policy at every update, the improvements over the estimations on the value functions can be applied more frequently. In the extreme case, a single iteration can be used to evaluate the policy, which results in the Value iteration method.

B.2.3 Value Iteration

As mentioned, the Value iteration approach updates the Bellman optimality equations at every evaluation step. Unlike Policy iteration, this method does not require to explicitly declare a policy, instead it directly optimizes the value function. Value iteration operates in the following way,

$$v_1 \rightarrow v_2 \rightarrow \dots v_k \dots \rightarrow v^*$$

$$v_{k+1}(s) \leftarrow \max_a \bar{\mathcal{R}}(s, a) + \gamma \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, a) \cdot v_k(s') \quad (\text{B.14})$$

B.3 Model-Free Learning: Value-based Learning

In model-free learning, opposed to model-based planning, the dynamics that define the model $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R} \rangle$ are not available. In this case, the agent discovers the model interacting with it, sampling trajectories from the environment. Model-free learning can be also applied for:

- **Prediction:** to discover how a policy π behaves in an unknown MDP. The goal is to discover the value functions v_{π} or q_{π} .
- Or **optimal control:** to achieve the optimal policy π_* only relying on the interaction with the problem.

B.3.1 Value-based Prediction

As mentioned, model-free prediction uses episodes of experiences on the model to evaluate a certain policy π . Depending on whether the complete or partial episodes are used to perform the evaluation, model-free learning can be classified in:

- **Monte-carlo learning:** which learns from complete episodes from experience.
- Or **Temporal difference learning:** performs updates based on single step backups.

Monte-Carlo Learning

Monte-Carlo (MC) learning operates over complete episodes of experience. Therefore, this method is only applicable to episodic MDPs, as episodes must terminate. To obtain the value functions, enough episodes under the policy π need to be collected,

$$\tau_\pi : \{S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T\} \sim \pi$$

The underlying idea behind this method is to estimate of the value functions using the empirical return observed during the experiences. Thus, computing the Bellman expectation equation is avoided.

$$G_t = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T \implies v_\pi(s) = \mathbb{E}_{\tau \sim \pi}[G_t | S_t = s]$$

In particular, for every state s , this method counts the number of visits $N(s)$ on the state and the cumulative returns obtained from the visits $S(s) \leftarrow S(s) + G_t$. Then, the value function is estimated averaging them, $V(s) = S(s)/N(s)$. By the law of large numbers, the estimated state value converges to the true value function as the number of visits to the state increases, $V(s) \rightarrow v_\pi(s)$ as $N(s) \rightarrow \infty$.

Alternatively, this average operation can be computed in an additive way. In that case, the estimated value function $V(s)$ is updated after each episode τ adding to the previous value the error term resulting from the current estimation. However, as tracking the number of number of visits produced at every state is not convenient or in some cases it cannot be done due to the size of the state space \mathcal{S} , this operation is usually substituted by updates performed with a learning rate α . Particularly, this results in computing the running mean (i.e. old episodes are forgotten in favor of the recent ones), which facilitates the computation.

$$V(S_t) \leftarrow V(S_t) + \frac{1}{N(S_t)}(G_t - V(S_t)) \approx V(S_t) + \alpha(G_t - V(S_t)) \quad (\text{B.15})$$

Temporal-Difference Learning

Temporal difference (TD) learning extends the use of model-free learning to non-episodic models, allowing to learn from incomplete episodes. This is done using *bootstrapping*, a technique that completes partial trajectories with an estimation of the onward return on the remaining episode. In particular, it updates the value function $V(S_t)$ towards partial estimations as seen in the Bellman equations. I.e., it uses the one-step estimated return $R_{t+1} + \gamma V(S_{t+1})$ instead of the current return G_t that MC learning obtains from complete episodes (Eq. B.15). Due to this reason, it is said that TD learning exploits the Markov property.

$$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t)) \quad (\text{B.16})$$

where

- $R_{t+1} + \gamma V(S_{t+1})$ is the TD target,
- and $R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$ is the TD error, also represented as δ_t .

The Bias/Variance trade-off: TD learning reduces the variance in the estimations when compared to MC learning. This is because in MC learning the variance is produced over the whole trajectory, whereas in TD learning only the stochasticity on the immediate step is included on the model. Due to this lower variance, TD learning is more efficient than MC learning. However, the TD target is a biased estimator of $v_\pi(s)$, which usually results in worst convergence properties.

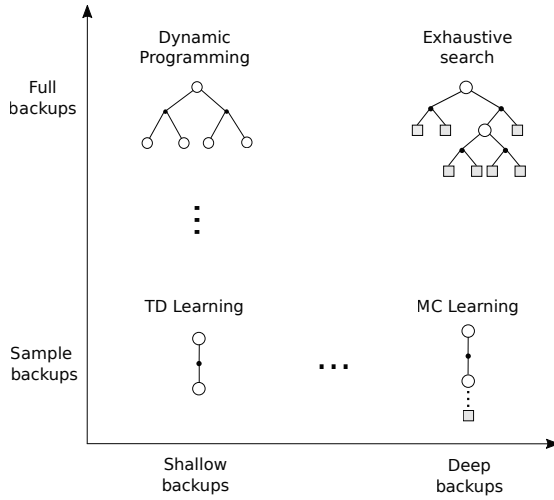


Fig. B.1: Comparison on RL methods. Two dimensions of RL algorithms, based on the backups used to learn or construct a policy are depicted. At the extremes: dynamic programming, exhaustive search, TD learning and Monte Carlo approaches. Bootstrapping extends from TD learning to n-step TD learning methods, with Monte Carlo approaches not relying on bootstrapping at all. Another possible dimension is the width of the backups, from TD learning and MC learning sample actions to traditional tree searches performed on a wider breadth. Recreated from [108].

The different techniques to perform backups in an MDP are summarized in Fig.B.3.1. The vertical axes represent the width of the backups. As observed, traditional DP or exhaustive search methods used full-width backups whereas RL methods are based on sampling trajectories. On the other hand, the backups that MC learning uses are deep, whereas TD learning only require shallow single-steps estimations.

Temporal-Difference (λ)

In order to control the bias/variance trade-off in the backups there exists several alternatives. A well-known approach is Temporal Difference (λ) or TD(λ). TD(λ) extends the idea of temporal difference further on multiple step backups, applying the bootstrapping concept to multiple steps in the trajectory. Particular cases of this technique are TD(0) and MC learning, which represent the extreme cases in the depth for this technique.

$$\begin{array}{ll}
 n=1 & \text{TD}(0) \quad G_t^{(1)} = R_{t+1} + \gamma V(S_{t+1}) \\
 n=2 & \quad \quad \quad G_t^{(2)} = R_{t+1} + \gamma R_{t+2} + \gamma^2 V(S_{t+2}) \\
 \vdots & \quad \quad \quad \vdots \\
 n = \infty & \text{MC} \quad \quad G_t^{(\infty)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T
 \end{array}$$

The n-step return is defined as,

$$G_t^{(n)} \doteq R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n V(S_{t+n}) \quad (\text{B.17})$$

The value function estimation using n-step TD learning is therefore obtained computing,

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t^{(n)} - V(S_t)) \quad (\text{B.18})$$

However, selecting the value of n is not trivial, as different values of it usually lead to significant different solutions. The solve to this problem, TD(λ) averages the n-steps returns over different steps, making it a more robust method to the choice of the λ parameter, as it combines information from several setups.

The λ return, G_t^λ , combines all n -step returns $G_t^{(n)}$ using a geometric weighed average over the n returns with a decay factor $(1 - \lambda) \cdot \lambda^{n-1}$,

$$G_t^{(\lambda)} \doteq (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} G_t^{(n)} \quad (\text{B.19})$$

Finally, the updates in the value function are performed as follows,

$$V(S_t) \leftarrow V(S_t) + \alpha(G_t^{(\lambda)} - V(S_t)) \quad (\text{B.20})$$

B.3.2 Value-based Control

Model-free control studies how to obtain the optimal policy π^* in an unknown MDP. To this end, the agent uses the trajectories sampled from the environment not only to explore it but also to improve its policy π .

In model-free control the agent faces a new dilemma, **the exploration-exploitation trade-off**. In the context of RL, this represents the trade-off between the need to obtain new knowledge from the environment and the need to exploit the acquired knowledge to improve performance of the model. I.e., exploration looks for the selection of actions that even though are not optimal according to the knowledge of the agent, may lead to better results; and exploitation, to enact the execution of actions that are optimal according to agent's current best estimate of the optimal policy π^* . During the training phase, the exploration is required to discover more about the optimal strategy, but the exploitation is also required to strengthen the policy π .

According to the experiences used to improve the policy π , there are two distinguished ways to approach optimal control:

- **On-policy** learning: the policy π is improved based on experience sampled from the same policy.
- **Off-policy** learning: the process is able to improve the policy π using experiences of a different policy μ .

On-policy Monte-Carlo Control

Monte-Carlo control uses Policy Iteration method (Annex B.2.2) to perform an iterative process in which the policy is improved. This iterative process consist in a:

- Monte-Carlo Policy Evaluation,
- followed by a ϵ -greedy Policy Improvement.

To this end, the policy π is evaluated using Monte-Carlo Policy evaluation. As mentioned, this strategy uses the whole trajectory to perform estimations over the value function. In this case, the action-value function Q_π is used as it enables to directly perform the Policy Improvement,

$$Q_\pi(S_t, A_t) \leftarrow Q_\pi(S_t, A_t) + \alpha(G_t - Q_\pi(S_t, A_t)) \quad (\text{B.21})$$

Once the value function Q_π is known, Policy Improvement is applied. Unlike in model-based learning where the improvement is done acting greedily over the action-value function, $\pi'(s) = \arg \max_{a \in \mathcal{A}} q_\pi(s, a)$, this cannot be applied in model-free learning. This is because the dynamics of the model are unknown and relying in non-representative explorations could

Algorithm 2 Sarsa Algorithm

```
1: procedure
2:   Initialize  $Q(s, a)$ ,  $\forall s \in \mathcal{S}, a \in \mathcal{A}(s)$  arbitrarily.
3:   repeat(for each episode):
4:     Set  $s$  to the initial state
5:     for each step in the episode do
6:       Take  $a$  and observe the reward  $r \sim R(s, a)$  and next state  $s'$ 
7:       Choose  $a'$  from  $s'$  using the policy derived from  $Q$  (e.g.  $\epsilon$ -greedy)
8:        $Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma Q(s', a') - Q(s, a))$ 
9:       Update  $s \leftarrow s'$  and  $a \leftarrow a'$ 
10:    end for
11:  until convergence
12: end procedure
```

lead to improvements in the policy that represent the best current performance instead of the optimal policy. Here, due to the exploration-exploitation dilemma, there is the need to obtain new knowledge from the model and use that knowledge to improve the policy. The goal of Reinforcement Learning can be seen as finding the right balance between exploration and exploitation to discover capable policies.

In order to incorporate exploration in the learning process, this method uses the ϵ -**greedy Policy Improvement** method. This simple technique relies in an ϵ -greedy action selection to perform exploration. This means that with a probability ϵ the agent chooses an arbitrary action and with probability $(1 - \epsilon)$, it acts greedy over the action-value function Q .

$$\pi(a|s) = \begin{cases} 1 - \epsilon & \text{then } \pi'(s) = \arg \max_{a \in \mathcal{A}} Q\pi(s, a) \\ \epsilon & \text{otherwise} \end{cases}$$

ϵ -greedy Policy Improvement presents asymptotic convergence to the best optimal policy only if the GLIE property is satisfied. For this method to be GLIE, the ϵ has to reduce proportionally to the number of steps $\epsilon_k \propto 1/k$. In that case, it can be proven that it converges to the optimal action-value function, $Q(s, a) \rightarrow q^*(s, a)$.

Definition B.3.1 A learning policy is called *GLIE (Greedy in the Limit with Infinite Exploration)* if it satisfies the following two properties:

- All state-action pairs are explored infinitely many times: $\lim_{k \rightarrow \infty} N_k(s, a) = \infty$
- The policy converges on a greedy policy:

$$\lim_{k \rightarrow \infty} \pi_k(s, a) = 1(a = \arg \max_{a' \in \mathcal{A}} Q_k(s, a'))$$

On-policy Temporal-Difference Control

As previously introduced in model-based learning, Temporal-difference (TD) learning presents several advantages over Monte-Carlo (MC) learning. Due to the bootstrapping technique, TD learning achieves lower variance and gains efficiency in the learning process. In this section, TD learning is applied for optimal on-policy control, this is the Sarsa algorithm (see Algorithm 2).

In the Sarsa algorithm, the value function is updated towards the estimation of the TD target. And the exploration is an ϵ -greedy action selection that it is combined with the improvement step. Formally, the Sarsa algorithm iterates over the following steps,

- Policy evaluation: TD Policy evaluation over the action-value function Q .

$$Q(s, a) \leftarrow Q(s, a) + \alpha(R + \gamma Q(s', a') - Q(s, a))$$

- Policy improvement: ϵ -greedy policy improvement.

This procedure can be extended for n-steps backups, as discussed in TD(λ). This results in the **Sarsa** (λ) algorithm. To this purpose, first the n-step Q-return shall to be defined,

$$Q_t^{(n)} = R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{n-1} R_{t+n} + \gamma^n Q(S_{t+n}, A_{t+n}) \quad (\text{B.22})$$

Following the same procedure as in Sec. B.3.1, the $Q^{(\lambda)}$ return is defined arithmetically combining all n-step Q-returns $Q_t^{(n)}$ to obtain a model that convenient account of all different n values.

$$Q_t^{(\lambda)} = (1 - \lambda) \sum_{n=1}^{\infty} \lambda^{n-1} Q_t^{(n)} \quad (\text{B.23})$$

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(Q_t^{(\lambda)} - Q(S_t, A_t)) \quad (\text{B.24})$$

TD(λ) improves the convergence, however it still partially suffers from some of the same disadvantages of MC-learning. E.g. the updates cannot be performed until the n-steps backups in the future are computed, which delays the updates. To solve this issue, a variant of TD(λ) exist that only relies in past backups. This algorithm is the **Backward-view TD** (λ) and it enables to perform online updates at every step t .

When relying on the past events a new dilemma surges, the credit assignment problem [79]. This problem arise from the impossibility to determine the past actions that had led to the current outcome. E.g., are frequent actions responsible of the current reward or just recent ones? To solve this issue a technique called Eligibility traces assigns credit to both, recent and frequent events dynamically during the roll-outs. Eligibility traces increases the credit to state-action pairs when they are visited, and decrease them exponentially when the agent does not. Thus, this method updates the value function $Q(s, a)$ for every state s and action a in proportion not only to the TD-error δ_t but also its frequency, this is the eligibility trace $E_t(s, a)$.

$$\begin{aligned} E_0(s, a) &= 0 \\ E_t(s, a) &= \gamma \lambda E_{t-1}(s, a) + 1(S_t = s, A_t = a) \end{aligned} \quad (\text{B.25})$$

The eligibility traces are stored for every recent state-action pair visited. Finally, the way this term is included in the update of the value function is weighing the TD-error,

$$\delta_t = R_{t+1} + \gamma Q(S_{t+1}, A_{t+1}) - Q(S_t, A_t) \quad (\text{B.26})$$

$$Q(s, a) \leftarrow Q(s, a) + \alpha \delta_t E_t(s, a) \quad (\text{B.27})$$

Algorithm 3 Sarsa (λ) Algorithm

```
1: procedure
2:   Initialize  $Q(s, a)$ ,  $\forall s \in \mathcal{S}, a \in \mathcal{A}(s)$  arbitrarily
3:   repeat(for each episode):
4:     Initialize the eligibility trace  $E(s, a) = 0, \forall s \in \mathcal{S}, a \in \mathcal{A}(s)$ 
5:     Set  $s$  to the initial state
6:     for each step in the episode do
7:       Take  $a$  and observe the reward  $r \sim R(s, a)$  and next state  $s'$ 
8:       Choose  $a'$  from  $s'$  using the policy derived from  $Q$  (e.g.  $\epsilon$ -greedy)
9:       Compute the TD-error:  $\delta \leftarrow r + \gamma Q(s', a') - Q(s, a)$ 
10:      Update the eligibility trace:  $E(s, a) \leftarrow E(s, a) + 1$ 
11:      for all  $s \in \mathcal{S}, a \in \mathcal{A}(s)$  do
12:         $Q(s, a) \leftarrow Q(s, a) + \alpha \delta E(s, a)$ 
13:         $E(s, a) \leftarrow \gamma \lambda E(s, a)$ 
14:      end for
15:      Update  $s \leftarrow s'$  and  $a \leftarrow a'$ 
16:    end for
17:  until convergence
18: end procedure
```

Off-policy Temporal-Difference Control

Algorithms that are characterized as off-policy perform updates using a separate behavior policy μ , independent of the policy being improved π . This is, the behavior policy μ is used to simulate trajectories over other policy without the need of executing them. A benefit of this technique is for example operate with a separately behavior policy, whereas the estimation policy can be deterministic.

To perform off-policy learning using estimations over the target policy π using a different policy distribution μ a common technique employed in statistics is Importance Sampling (IS).

Definition B.3.2 *Importance Sampling (IS) technique estimates the expected value of a function $f(x)$ from data distribution $P(x)$, relying on samples of a different distribution $Q(x)$.*

$$\begin{aligned} \mathbb{E}_{x \sim P}[f(x)] &= \sum P(x) f(x) \\ &= \sum Q(x) \frac{P(x)}{Q(x)} f(x) \\ &= \mathbb{E}_{x \sim Q} \left[\frac{P(x)}{Q(x)} f(x) \right] \end{aligned} \tag{B.28}$$

For the purpose of RL, importance sampling can be applied to estimate the total return generated by the target policy π using the actual return obtained over a different distribution μ . This is achieved as follows,

$$G_t^{\pi/\mu} = \frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} \frac{\pi(A_{t+1}|S_{t+1})}{\mu(A_{t+1}|S_{t+1})} \dots \frac{\pi(A_T|S_T)}{\mu(A_T|S_T)} G_t \tag{B.29}$$

However, a major problem of IS is that it dramatically increases variance. Because of this reason, IS is commonly used for off-Policy TD-learning, as it is only required to estimate a single correction in over the TD-error of the next step. For this method to work, policies need to be similar. Formally,

$$V(S_t) \leftarrow V(S_t) + \alpha \left(\frac{\pi(A_t|S_t)}{\mu(A_t|S_t)} (R_{t+1} + \gamma V(S_{t+1})) - V(S_t) \right) \tag{B.30}$$

Algorithm 4 Q-Learning Algorithm

```

1: procedure
2:   Initialize  $Q(s, a)$ ,  $\forall s \in \mathcal{S}, a \in \mathcal{A}(s)$  arbitrarily, and  $Q(\text{terminating state}, \cdot) = 0$ 
3:   repeat(for each episode):
4:     Set  $s$  to the initial state
5:     for each step in the episode do
6:       Choose  $a$  from  $s$  using policy derived from  $Q$  (e.g.  $\epsilon$ -greedy)
7:       Take  $a$  and observe the reward  $r \sim R(s, a)$  and next state  $s'$ 
8:        $Q(s, a) \leftarrow Q(s, a) + \alpha(r + \gamma \max_a Q(S', a) - Q(s, a))$ 
9:       Update  $s \leftarrow s'$ 
10:    end for
11:  until convergence
12: end procedure

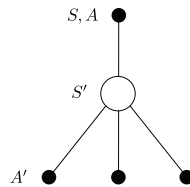
```

A well-known algorithm that uses off-policy TD learning achieving outstanding results is **Q-learning** (Algorithm 4). Unlike off-policy TD-learning, Q-learning updates the Q-values without requiring of IS. Q-learning uses a target policy π is greedily w.r.t. $Q(s, a)$, $\pi(S_{t+1}) = \arg \max_{a'} Q(S_{t+1}, a')$, and the behaviour policy μ is used to perform the exploration, e.g., ϵ -greedily. Q-learning learns therefore a greedy behaviour from an stochastic exploratory policy.

The Q-Learning target then simplifies:

$$\begin{aligned}
 Q_{target} &= R_{t+1} + \gamma Q(S_{t+1}, a') \\
 &= R_{t+1} + \gamma Q(S_{t+1}, \arg \max_{a'} Q(S_{t+1}, a')) \\
 &= R_{t+1} + \max_{a'} \gamma Q(S_{t+1}, a')
 \end{aligned} \tag{B.31}$$

The Q-learning control algorithm is proved to converge to the optimal action-value function: $Q(s, a) \rightarrow q^*(s, a)$. The update operates as follows,



$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha(R + \gamma \max_{a'} Q(S', a') - Q(S_t, A_t)) \tag{B.32}$$

B.4 Model-Free Learning: Policy-based Learning

In contrast to value-based methods, policy optimization does not need to estimate a value function, but directly search for an optimal policy π^* . To this end, the policy is directly parameterized by $\theta \in \mathbb{R}^N$, and the goal is to optimize it π_θ for achieving the maximum expected reward J_π . Formally,

$$\max_{\theta} J_\pi(\theta) = \max_{\theta} \mathbb{E} \left[\sum_{t=0}^T R(S_t, A_t) | \pi_\theta \right] \quad (\text{B.33})$$

Policy optimization methods directly output the parameters for a probability distribution; for continuous actions, this could be the mean and standard deviations of Gaussian distributions, whilst for discrete actions this could be the individual probabilities of a multinomial distribution. This method relies on the stochasticity of the policy $\pi_\theta(a|s)$ to perform exploration. In order to act, it samples actions from the output distribution and based on the result obtained, the model directly optimize its policy. The goal in policy optimization is to explore diverse behaviours and foster the model to converge towards the positive ones.

Policy optimization present some advantages when compared to value-base methods. They have a better convergence, as the policy is directly optimized. Also, they are compatible with in high-dimensional or continuous action spaces. And finally, can converge to stochastic policies. On the other hand, these methods typically converge to a local rather than a global optimum. Also present higher variance and are less sample-efficient. Policy optimization algorithms are on-policy by design. Consequently, samples collected on previous update iterations become useless for further steps on the learning process. This is the key reason why policy gradient algorithms are usually less sample-efficient than value-based methods.

To directly optimize the objective function $J_\pi(\theta)$, several approaches have been used in the literature. These can be classified into: gradient-free methods (e.g., cross entropy method [109], genetic algorithms [106], etc) and gradient-based methods (gradient descent, conjugate gradient, quasi-newton methods, etc). However, as gradient-free optimization are only effective in low-dimensional parameter spaces, this makes gradient-based training the method of choice for most deep RL algorithms [38].

B.4.1 Policy Gradients

Policy Gradients (PG) algorithm, provides a strong learning mechanism on how to optimize the parameterized policy π_θ . In order to estimate the expected return J_π , this method averages over multiple sampled trajectories executed over the current policy. Thus, sampling induces this problem into a stochastic approximation process.

Let us denote τ the trajectory of state-action sequences that conform an episode of length T ,

$$\tau_\pi : \{S_0, A_0, R_1, \dots, S_{T-1}, A_{T-1}, R_T, S_T\} \sim \pi$$

and $R(\tau)$ the reward obtained in the trajectory τ ,

$$R(\tau) = \sum_{t=0}^{T-1} \gamma^t R_{t+1} \quad (\text{B.34})$$

The goal in policy optimization is to find the parameters θ that maximize the expected accumulated reward the agent achieves over all possible trajectories,

$$\max_{\theta} J(\theta) = \max_{\theta} \mathbb{E}[R(\tau)] = \max_{\theta} \sum_{\tau} \mathcal{P}(\tau|\theta) \cdot R(\tau) \quad (\text{B.35})$$

To optimize this objective, PG performs a gradient ascent on the policy parameters: $\theta \leftarrow \theta + \alpha \nabla_{\theta} J(\theta)$. However, this is not directly computed, as the expression above depends on the dynamics of the model (which are unknown). In particular, $\mathcal{P}(\tau)$ represents the probability of performing a trajectory given a policy behaviour. PG decouples the gradient on this objective function from the environment dynamics, enabling a model-free learning. To achieve this, this method uses the "log-derivative" trick to insert the gradient into the expectation equation (B.35). Taking the gradient of J_{π} w.r.t. θ ,

$$\begin{aligned} \nabla_{\theta} J(\theta) &= \nabla_{\theta} \sum_{\tau} \mathcal{P}(\tau|\theta) \cdot R(\tau) \\ &= \sum_{\tau} \nabla_{\theta} \mathcal{P}(\tau|\theta) \cdot R(\tau) \\ &= \sum_{\tau} \mathcal{P}(\tau|\theta) \cdot \frac{\nabla_{\theta} \mathcal{P}(\tau|\theta)}{\mathcal{P}(\tau|\theta)} \cdot R(\tau) \\ &= \sum_{\tau} \mathcal{P}(\tau|\theta) \cdot \nabla_{\theta} \log \mathcal{P}(\tau|\theta) \cdot R(\tau) \end{aligned} \quad (\text{B.36})$$

In order to compute the gradient on the probability of the trajectory $\mathcal{P}(\tau)$, the path is decomposed into a chain of probabilities by the MDP assumption, whereby the next action only depends on the current state, and the next state only depends on the current state and action. Explicitly, taking gradients on this expression, the dynamics term removes from the equation.

$$\begin{aligned} \nabla_{\theta} \log \mathcal{P}(\tau|\theta) &= \nabla_{\theta} \left[\prod_{t=0}^{T-1} \mathcal{P}(S_{t+1}|S_t, A_t) \cdot \pi_{\theta}(A_t|S_t) \right] \\ &= \nabla_{\theta} \left[\sum_{t=0}^{T-1} \log \mathcal{P}(S_{t+1}|S_t, A_t) + \sum_{t=0}^{T-1} \log \pi_{\theta}(A_t|S_t) \right] \\ &= \nabla_{\theta} \sum_{t=0}^{T-1} \log \pi_{\theta}(A_t|S_t) \\ &= \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(A_t|S_t) \end{aligned} \quad (\text{B.37})$$

Finally, substituting the gradient on the trajectory into Eq. (B.36) the objective simplifies in the following expression (B.38). It is worth to mentioned that this equation is still an unbiased estimator of the original objective (B.35).

$$\nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}}[R(\tau)] = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\left(\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(A_t|S_t) \right) \cdot R(\tau) \right] \quad (\text{B.38})$$

Intuitively, PG work in the following way, it increases the likeliness of paths that achieve positive reward and decrease the probability of paths with negative reward.

B.4.2 Baseline

In PG method, the variance of the gradient estimations is sometimes excessive. This is because complete trajectories are used to perform this estimation. E.g., after one training batch one may exhibit a wide range of results for a trajectory: much better performance, equal performance, or worse performance. This variance in the results produce a slow and unreliable convergence on the optimization process. Due to this reason, there has been so much effort devoted to develop variance reduction techniques that mitigate this effect.

A common method to reduce the variance of the above expression is to introduce a baseline function $b \rightarrow \mathbb{R}$ in the reward term. This reduces the difference in the rewards while still maintains an unbiased estimator, as argued in (Williams et al., 1992) [122].

$$\nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau)] = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\left(\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(A_t | S_t) \right) \cdot (R(\tau) - b) \right] \quad (\text{B.39})$$

To demonstrate that with the inclusion of the baseline b , the gradient estimate remains unbiased, let us distribute and rearrange the equation to get

$$\nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau)] = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(A_t | S_t) \cdot R(\tau) - \sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(A_t | S_t) \cdot b \right] \quad (\text{B.40})$$

Due to the linearity of the expectation, the baseline term can be isolated. In the following, we demonstrate that the baseline term b does not modifies the gradient. This justifies that its inclusion does not cause bias to the original equation.

$$\begin{aligned} \mathbb{E}_{\tau \sim \pi_{\theta}} [\nabla_{\theta} \log \mathcal{P}(\tau | \theta) \cdot b] &= \sum_{\tau} \mathcal{P}(\tau | \theta) \cdot \nabla_{\theta} \log \mathcal{P}(\tau | \theta) \cdot b \\ &= \sum_{\tau} \mathcal{P}(\tau | \theta) \frac{\nabla_{\theta} \mathcal{P}(\tau | \theta)}{\mathcal{P}(\tau | \theta)} \cdot b \\ &= \sum_{\tau} \nabla_{\theta} \mathcal{P}(\tau | \theta) \cdot b \\ &= \nabla_{\theta} \left(\sum_{\tau} \mathcal{P}(\tau | \theta) \cdot b \right) = b \times 0 \end{aligned} \quad (\text{B.41})$$

The last thing to cover is why the introduction of the baseline reduces variance. To this purpose, let us analyze the variance,

$$\begin{aligned} &\text{Var} \left(\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(A_t | S_t) \cdot (R(\tau) - b) \right) \approx \\ &\approx \sum_{t=0}^{T-1} \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\left(\nabla_{\theta} \log \pi_{\theta}(A_t | S_t) \cdot (R(\tau) - b) \right)^2 \right] \approx \\ &\approx \sum_{t=0}^{T-1} \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\left(\nabla_{\theta} \log \pi_{\theta}(A_t | S_t) \right)^2 \right] \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\left(R(\tau) - b \right)^2 \right] \end{aligned} \quad (\text{B.42})$$

The optimization of the variance is a least squares problem, whose optimal solution occurs for the baseline b taking the value of $R(\tau)$. However, this exact value cannot be determined a priori. For to this reason, it is usually approximated to the expected return, $b \approx \mathbb{E}[R(\tau)]$.

Causality principle

The causality principle defines that future actions do not change past decision. In other words, present actions only impact the future. This can also be reflected into the objective function. In addition, removing the terms that do not depend on the current action from the objective equation helps to lower its variance. Thus, the causality principle is commonly used to define the objective function.

The causality principle empirically works well combined with the decay factor. The effect of both combined implies that an action at a time-step t , does only present an impact at the current time-step and decays several time-steps later. Thus, the gradient on the objective can be rewritten as,

$$\begin{aligned}
\nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}}[R(\tau)] &= \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\left(\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(A_t | S_t) \right) \cdot (R(\tau) - b) \right] \\
&= \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\left(\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(A_t | S_t) \right) \cdot \left(\left(\sum_{t=0}^{T-1} \gamma^t R_{t+1} \right) - b \right) \right] \\
&= \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(A_t | S_t) \cdot \left(\left(\sum_{t'=0}^{t-1} \gamma^{t'-t} R_{t'+1} + \sum_{t'=t}^{T-1} \gamma^{t'-t} R_{t'+1} \right) - b \right) \right] \quad (\text{B.43}) \\
&\approx \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(A_t | S_t) \cdot \left(\left(\sum_{t'=t}^{T-1} \gamma^{t'-t} R_{t'+1} \right) - b \right) \right] \\
&= \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(A_t | S_t) \cdot (G_t - b) \right]
\end{aligned}$$

In this case, the optimal baseline can be approximated by the state-dependent expected return $b(S_t) : \mathcal{S} \rightarrow \mathbb{R}$, which results in the expected value function for every state $v_{\pi}(S_t)$.

$$b(S_t) = \mathbb{E}(R_{t+1} + \gamma R_{t+2} + \dots + \gamma^{T-1} R_T) = \mathbb{E}_{\pi}[G_t | S_t] = v_{\pi}(S_t) \quad (\text{B.44})$$

The resulting equation increases the probability of an action proportionally to how much the returns at each time-step is better than the expected value function under the current policy. In order to implement this method, the state dependent baseline can be approximated using a separate function approximator $V_{\phi}^{\pi}(S_t)$, with parameters ϕ .

$$\begin{aligned}
\nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}}[R(\tau)] &= \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(A_t | S_t) \cdot (G_t - v_{\pi}(S_t)) \right] \\
&\approx \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(A_t | S_t) \cdot (G_t - V_{\phi}^{\pi}(S_t)) \right] \quad (\text{B.45})
\end{aligned}$$

Notice that to learn the value function approximator $V_{\phi}^{\pi}(S_t)$, the optimal labels are obtained at the end of the episode. Thus, supervised learning techniques can be used for achieving the Policy evaluation (e.g., a common technique is to optimize the Mean Square Error). The implementation of this technique can be seen in greater detail in Algorithm 5.

Algorithm 5 Vanilla Policy Gradients with state-dependent baseline

```
1: procedure
2:   Initialize the policy parameters  $\theta_0$  and the value function parameters  $\phi_0$ .
3:   repeat(for iteration=1,2,...):
4:     Collect a batch of  $B$  trajectories by executing the current policy:  $\tau_\pi$ 
5:     for each step  $t$  in the episode do
6:       Compute the return  $G_t$  and
7:       the advantage estimate  $A_t = G_t - V_\phi(S_t)$ 
8:     end for
9:     Estimate gradient:  $\hat{g}_\theta = 1/B \cdot \sum_{b=1}^B \sum_{t=1}^T (G_t^{(b)} - V_\phi^{(b)}(S_t)) \cdot \nabla_\theta \log \pi_\theta(A_t^{(b)} | S_t^{(b)})$ 
10:    Update the policy:  $\theta \leftarrow \theta + \alpha \hat{g}_\theta$ 
11:    Update baseline by minimizing:  $L_\phi = 1/B \cdot \sum_{b=1}^B \sum_{t=1}^T \|G_t^{(b)} - V_\phi^{(b)}(S_t)\|^2$ 
12:  until convergence
13: end procedure
```

B.5 Actor-Critic methods

Actor-critic (AC) methods combine the benefits of direct policy optimization with value function approximation in order to take advantage from both learning strategies. In particular, these methods have a separate structure to explicitly represent the policy independent of the return estimator. This architecture enables them to compute updates in the policy based on estimations done over incomplete roll-outs, which speeds up the learning process.

In AC methods, the policy structure is known as the actor, because it is the one that select actions; and the estimated value function is known as the critic, because it evaluates the actions made by the actor. The learning process is always on-policy, the critic must learn about and critique whatever policy is currently being followed by the actor. This is, the critic estimates the return and the actor uses it to weight the action distribution.

In order to build these models, usually two independent function approximators are used: the policy π_θ function for representing the actor, and a value function estimator (usually $V_\phi^\pi(s)$ or $Q_\phi^\pi(s, a)$, depending on the variant) with the parameters ϕ for the critic.

Q Actor-Critic (QAC)

AC extends the idea of estimating the return using a value function instead of relying on the empirical return G_t obtained at the end of the episode. In the case of Q Actor-critic or QAC, the critic estimates the action-value function to obtain a representation on the average return the actor obtains under its policy, $Q^\pi(s, a) = \mathbb{E}_{\tau \sim \pi_\theta} [G_t | S_t = s, A_t = a]$. Relying on the average return instead on a single trajectory reduces the variance experienced in PG. However, it also turns ACs into a biased estimator, as they rely on function approximators for this purpose.

$$\nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta} [R(\tau)] = \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(A_t | S_t) \cdot Q^\pi(S_t, A_t) \right] \quad (\text{B.46})$$

The critic uses policy evaluation to estimate the value function under the policy. In this case, it is done using MC learning. The resulting signal drives the learning process of both agents, the actor and the critic itself.

Algorithm 6 Q Actor-Critic (QAC)

```
1: procedure
2:   Initialize the parameters of the actor  $\theta$  and the critic  $\phi$ .
3:   repeat(for each episode):
4:     Initialize  $s$  to the initial state and sample  $a \sim \pi_\theta(a|s)$ 
5:     for each step in the episode do
6:       Take  $a$  and observe the reward  $r \sim R(s, a)$  and next state  $s'$ 
7:       Then sample the next action  $a' \sim \pi_\theta(\cdot|s')$ 
8:       Update the policy parameters:  $\theta \leftarrow \theta + \alpha \nabla_\theta \log \pi_\theta(a|s) Q_\phi(s, a)$ 
9:       Compute the TD-error:  $\delta \leftarrow r + Q_\phi(s', a') - Q_\phi(s, a)$ 
10:      and use it to update the critic:  $\phi \leftarrow \phi + \beta \delta \nabla_\phi Q_\phi(s, a)$ 
11:      Update  $s \leftarrow s'$  and  $a \leftarrow a'$ 
12:    end for
13:  until convergence
14: end procedure
```

Advantage Actor-Critic (A2C)

The baseline technique can also be used in AC methods to reduce the variance. As mentioned, usually the state value function $V^\pi(s)$ is used as the baseline estimator. This it is subtracted from the $Q^\pi(s, a)$ function. Intuitively, this gives a new quantity that indicates how much better is a certain action at that state compared to the average. This quantity is the advantage function and is defined as $A^\pi(s, a) = Q^\pi(s, a) - V^\pi(s)$. This leads to the A2C algorithm with the gradient update,

$$\begin{aligned} \nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)] &= \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(A_t|S_t) \cdot (Q^\pi(S_t, A_t) - V^\pi(S_t)) \right] \\ &= \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(A_t|S_t) \cdot A^\pi(S_t, A_t) \right] \end{aligned} \quad (\text{B.47})$$

A2C method uses the critic to estimate the advantage directly, instead of both value functions separately.

TD Actor-Critic

The advantage can be also approximated by the TD error. Incorporating TD-learning to the valued function estimation may be helpful especially if one want to update the policy after each transition. Formally,

$$\begin{aligned} A^\pi(S_t, A_t) &= R_{t+1} + \gamma V^\pi(S_{t+1}) - V^\pi(S_t) = \delta(S_t, A_t) \\ \nabla_\theta \mathbb{E}_{\tau \sim \pi_\theta}[R(\tau)] &= \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(A_t|S_t) \cdot (R_{t+1} + \gamma V^\pi(S_{t+1}) - V^\pi(S_t)) \right] \\ &= \mathbb{E}_{\tau \sim \pi_\theta} \left[\sum_{t=0}^{T-1} \nabla_\theta \log \pi_\theta(A_t|S_t) \cdot \delta_t(S_t, A_t) \right] \end{aligned} \quad (\text{B.48})$$

Generalized Advanced Estimator (GAE)

Actor-Critic is an on-policy algorithm by design, hence bootstrapping to multiple-steps can be performed. Instead of relying in a single time-step approximations ($N = 1$) as previously shown in TD Actor-critic, when the approximation is built over multiple N steps, the trade-off between bias and variance can be controlled (see Sec. B.3.1). However, performing an advantage estimation using N steps of interaction after given state-action pair presents also some difficulties. Usually, finding a good value for N as hyperparameter is difficult as its «optimal» value may change throughout the learning process.

$$\begin{aligned}
 A_t^{(1)} &= R_{t+1} + \gamma V^\pi(S_{t+1}) - V^\pi(S_t) \\
 A_t^{(2)} &= R_{t+1} + \gamma R_{t+2} + \gamma^2 V^\pi(S_{t+2}) - V^\pi(S_t) \\
 &\dots \\
 A_t^{(\infty)} &= R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots - V^\pi(S_t)
 \end{aligned} \tag{B.49}$$

In Generalized Advantage Estimation (GAE) [99] it is proposed to use TD- (λ) to ensemble out of different n-step advantage estimators to smooth the weight of the different steps in the final solution and thus, facilitate the selection of the λ hyperparameter. The GAE advantage estimator defines as,

$$A^{GAE(\gamma, \lambda)}(S_t, A_t) = (1 - \lambda)(A_t^{(1)} + \lambda A_t^{(2)} + \lambda^2 A_t^{(3)} + \dots) = \sum_{l=0}^{\infty} (\gamma \lambda)^l \cdot \delta(S_{t+l}, A_{t+l})$$

The GAE method constructs the advantage as an exponentially-decayed sum of the residual terms. The above formula describes the estimator GAE (γ, λ) for $\lambda \in [0, 1]$ where adjusting λ allows an smooth control over bias-variance trade-off: $\lambda \rightarrow 0$ corresponds to Actor-Critic with higher bias and lower variance while $\lambda \rightarrow 1$ corresponds to a Monte-Carlo PG with no bias and high variance.

Lastly, γ is the discount factor, and it also adjusts the bias-variance tradeoff but in a different way. Controlling the importance of the current state over future states. Henceforth, both parameters γ and λ , control different mechanisms that in the end contribute to the same behaviour.

$$\begin{aligned}
 A^{GAE(\gamma, 0)}(S_t, A_t) &= \delta(S_t, A_t) \\
 A^{GAE(\gamma, 1)}(S_t, A_t) &= \sum_{l=0}^{\infty} \gamma^l \cdot \delta(S_{t+l}, A_{t+l}) = \sum_{l=0}^{\infty} \gamma^l R_{t+1+l} - V(S_{t+l}) = G_t - V(S_t)
 \end{aligned} \tag{B.50}$$

Finally, the gradient update is expressed as:

$$\nabla_{\theta} \mathbb{E}_{\tau \sim \pi_{\theta}} [R(\tau)] = \mathbb{E}_{\tau \sim \pi_{\theta}} \left[\sum_{t=0}^{T-1} \nabla_{\theta} \log \pi_{\theta}(A_t | S_t) \cdot A^{GAE(\gamma, \lambda)}(S_t, A_t) \right] \tag{B.51}$$

In this appendix C, supplementary information on the Job Shop Scheduling Problem (JSP) is presented. Particularly, (1) details on the heuristic and metaheuristic algorithms included in the experimental study, (2) specifications on the implementation of the RL architectures for the JSP, and (3) running times of the learning process are introduced.

C.1 Heuristic and metaheuristic algorithms for the Job Shop Problem

Conducted experimental study in Section 5, compared the performance of the proposed model with some of the most representative heuristics and metaheuristics for the JSP in the literature. In the following, we present a summary of the four heuristics algorithms included, yet more information about them can be found in [71].

- **Shortest Processing Time (SPT):** it is one of the most used heuristics for solving the JSP problem. At each iteration, it selects the job with the least processing time from the competing list and schedules it ahead of the others. With illustrative purposes, let us considerate that two operations of different jobs are competing at a time-step for the same machine to be released. In that case, the operation with the shortest processing time will be scheduled first.
- **Longest Processing Time (LPT):** it follows the opposite rule of the SPT heuristic. The operation with the longest processing time is scheduled ahead of the competitors.
- **First-Come-First-Served (FCFS):** this rule schedules the jobs simply in the order of job arrival. There is no consideration on the processing time or any other information.
- **Least Work Remaining (LWR):** it is also an extension of SPT, this rule dictates the operation to be scheduled according to the processing time remaining before the job is completed. The less work remaining in a job, the earlier it is scheduled.

In addition to the classical heuristic algorithms exposed above, a metaheuristic, particularly, a Genetic Algorithm (GA) [8] has been included in the experimental study. The implementation corresponds to [123]. Regarding the hyperparameter setting, a population of 300 individuals, a crossover rate of 0.8 and a mutation rate of 0.3 were set. Finally, the algorithm was run 500 generations before stopping (enough iterations to converge in the different problems included in the study).

C.2 Implementation details

This appendix complements the details on the neural model introduced in Section 5.1.1. The proposed model presents two different input sources: the instance of the problem s , which is defined by the M and D feature matrices, and the state of the environment d_t , represented by the state of the machines and the time for the previous operations to finish. In order to embed

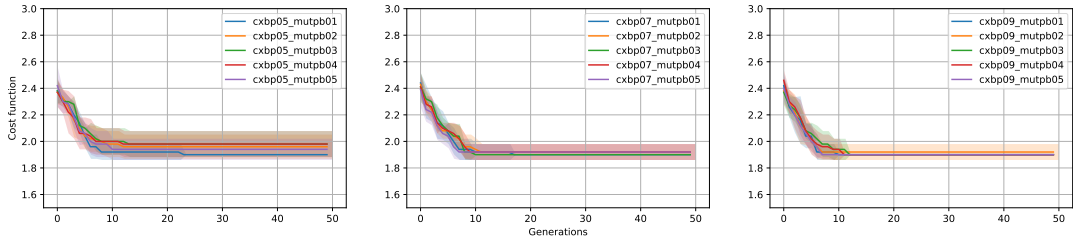


Fig. C.1.: Tuning process in the Genetic Algorithm.

both sources, single linear layers with a vector size of 64 are utilized. We find that normalizing the input vectors and embedding them in a higher feature space yields to superior solutions.

In regard with the details on the architecture, the RNN encoder used to codify the sequences of operations for each job is a single LSTM [45] layer with a hidden state size of 64. Specifically, it is a unidirectional encoder working backwards. This manner, the encoder outputs the codification of the remaining operations for a job, starting at every point in the sequence. This procedure is computed once and stored to be used during the interaction with the environment. In that process, an index vector i_t points at the current operation to be scheduled and the feature vector e_{ij} is gathered for each job to create the context vector c_t .

Lastly, the DNN decoder consists in multiple dense layers with a ReLU activation. The variables are initialized with Xavier initialization [46]. The batch size is 800, and it is formed by 20 different instances introduced 40 times each. This is done to perform the Reinforce with self-competing baseline described in this work (more detailed information available in Appendix 4.4.3). The optimizer is Adam [63] with a learning rate of $5 \cdot 10^{-4}$. The gradients are clipped to the norm by a value of 1, and a dropout with a probability of 0.1 is used in the LSTM encoder.

C.3 Run times

The code for the RL model proposed in the work is implemented in PyTorch¹ [86]. The model entirely run on a GPU, i.e. both the environment and the agent are implemented as tensor operations. This allows to fully parallelize the process, executing the whole batch operations at once. Even though current RL frameworks (e.g. OpenAI baselines [37]) allow to execute the environment in parallel threads using multiple CPUs, this approach permits to significantly reduce the learning time. In order to train the model a single GPU (2080Ti) was used. The times required to perform a single epoch are described below:

Tab. C.1.: Computation time per epoch required by the RL model in the JSP problem.

	JSP10x10	JSP15x15	JSP20x20	JSP25x25
$\lambda = 0$	2.2s	4.7s	7.8s	11.5s
$\lambda = 1$	2.5s	5.4s	9.9s	12.7s

The datasets used in the experimentation are included along the code. The instances have been created following the OR-Library [14] format². For every instance, there is a heading that indicates the number of jobs n and the number of machines m . Then, there is a one line for each

¹Code will be available in: (Link)

²Taillard instances for scheduling problems[110]: (Link)

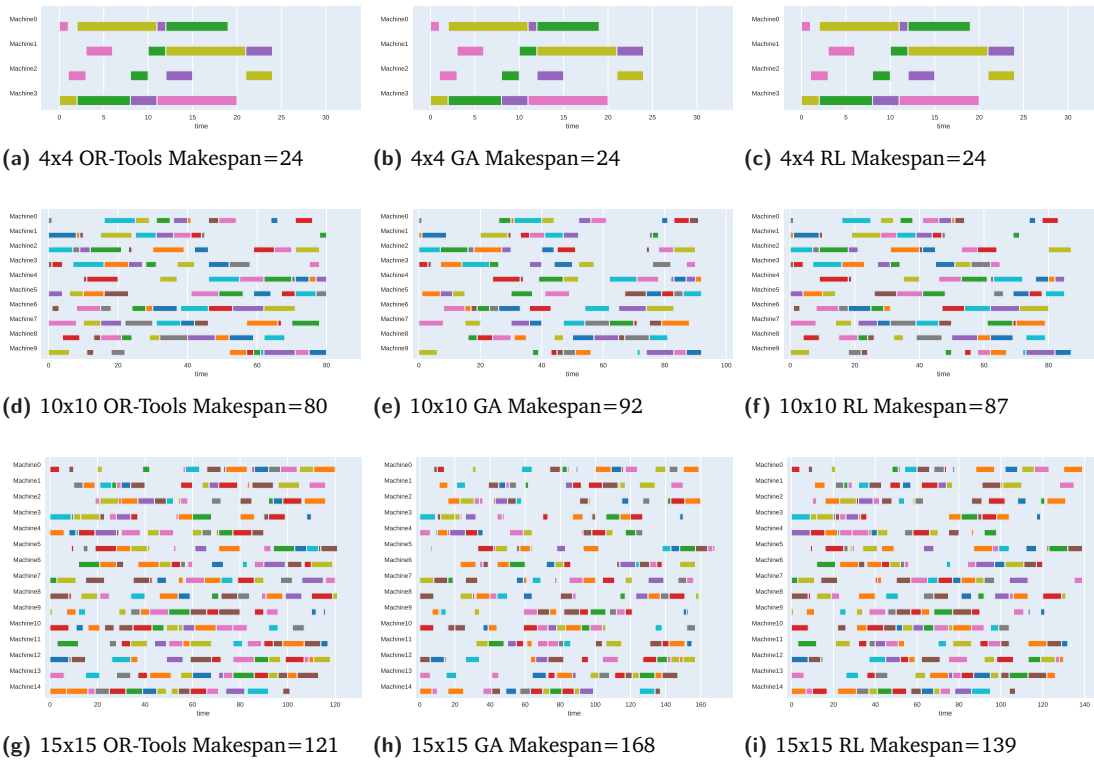


Fig. C.2.: Example solutions for the classic JSP ($\lambda = 0$) problem, Gantt diagrams. On the left, the optimal solutions computed with OR-Tools; on the center, the result of the Genetic Algorithm and on the right, the results obtained by the RL model with a sampling technique.

job, listing the machine number and processing time for each operation. The results provided in the experimentation are obtained after performing a training of 4000 epochs on those datasets.

Finally, to visualize the results obtained by the different alternatives, a comparison of the solutions presented as Gantt diagrams is also included. This is done for the classic JSP (Figure C.2) and for the *no idle time* variant (Figure C.3). Although in the figures a strategy cannot be seen at a glance, the RL model infers a competitive policy. This policy cannot be predicted, and guarantees in the results cannot be given. Yet a consistency in the results is observed.

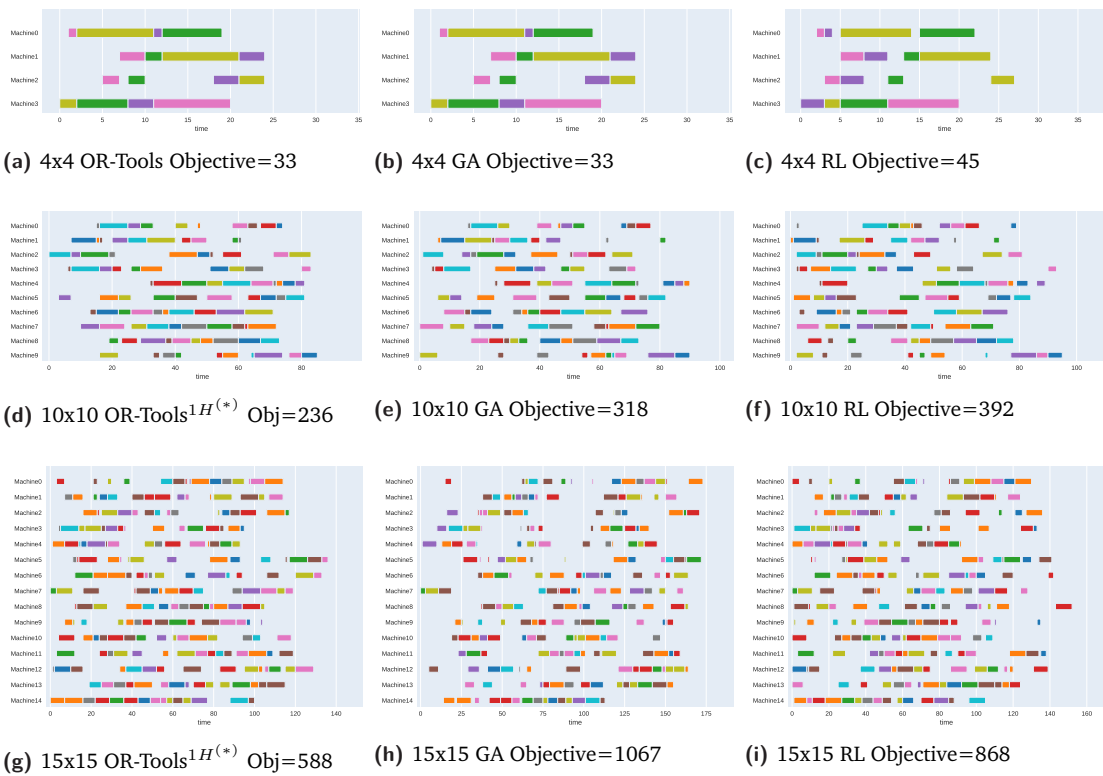


Fig. C.3.: Example solutions for the JSP with limited idle time ($\lambda = 1$) problem, Gantt diagrams. On the left, the solutions computed with OR-Tools; on the center, the result of the Genetic Algorithm and on the right, the results obtained by the RL model with a sampling technique.

Notation summary

$\mathbf{x} \in \mathcal{X}$	Problem space
$\mathbf{y} \in \mathcal{Y}$	Combinatorial space
$s \in \mathcal{S}$	States
$a \in \mathcal{A}$	Actions
$r \in \mathcal{R}$	Rewards signal
$c \in \mathcal{C}$	Constraint dissatisfaction signal
G	Episodic return
$\mathcal{P}(s' s, a)$	Transition probability distribution
$\pi(a s)$	Stochastic policy
$\pi^*(a s)$	Optimal policy
$V^\pi(s)$	State-value function following the policy π
$Q^\pi(s, a)$	Action-value function following the policy π
$V^*(s)$	Optimal state-value function
$Q^*(s, a)$	Optimal action-value function
$A(s, a)$	Advantage function
$b(s)$	Baseline function
α	Learning rate
γ	Discount factor
λ	Lagrange multiplier
β	Entropy regularization parameter
ϵ	PPO clipping parameter
η	Intrinsic reward parameter
ν	Value function loss parameter

Bibliography

- [1]Martín Abadi, Paul Barham, Jianmin Chen, et al. „Tensorflow: A system for large-scale machine learning“. In: *12th {USENIX} Symposium on Operating Systems Design and Implementation ({OSDI} 16)*. 2016, pp. 265–283 (cit. on p. 44).
- [2]Joshua Achiam, David Held, Aviv Tamar, and Pieter Abbeel. „Constrained policy optimization“. In: *arXiv preprint arXiv:1705.10528* (2017) (cit. on p. 35).
- [3]Bernardetta Addis, Dallah Belabed, Mathieu Bouet, and Stefano Secci. „Virtual Network Functions placement and routing optimization“. In: *CloudNet*. 2015, pp. 171–177 (cit. on p. 76).
- [4]Zafarali Ahmed, Nicolas Le Roux, Mohammad Norouzi, and Dale Schuurmans. „Understanding the impact of entropy on policy optimization“. In: *arXiv preprint arXiv:1811.11214* (2018) (cit. on p. 47).
- [5]Eitan Altman. *Constrained Markov decision processes*. Vol. 7. CRC Press, 1999 (cit. on p. 34).
- [6]Alejandro Marcos Alvarez, Quentin Louveaux, and Louis Wehenkel. „A supervised machine learning approach to variable branching in branch-and-bound“. In: *IN ECML*. Citeseer. 2014 (cit. on p. 11).
- [7]Thomas Anthony, Zheng Tian, and David Barber. „Thinking fast and slow with deep learning and tree search“. In: *Advances in Neural Information Processing Systems*. 2017, pp. 5360–5370 (cit. on p. 92).
- [8]Thomas Bäck, David B Fogel, and Zbigniew Michalewicz. *Handbook of evolutionary computation*. CRC Press, 1997 (cit. on pp. 84, 124).
- [9]Adrià Puigdomènech Badia, Bilal Piot, Steven Kapturowski, et al. „Agent57: Outperforming the atari human benchmark“. In: *arXiv preprint arXiv:2003.13350* (2020) (cit. on p. 99).
- [10]Adrià Puigdomènech Badia, Pablo Sprechmann, Alex Vitvitskyi, et al. „Never Give Up: Learning Directed Exploration Strategies“. In: *arXiv preprint arXiv:2002.06038* (2020) (cit. on pp. 51, 99).
- [11]Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. „Neural Machine Translation by jointly learning to align and translate“. In: *arXiv:1409.0473* (2014) (cit. on pp. 20, 31).
- [12]Radu Baltean-Lugojan, Pierre Bonami, Ruth Misener, and Andrea Tramontani. „Selecting cutting planes for quadratic semidefinite outer-approximation via trained neural networks“. In: (2018) (cit. on p. 11).
- [13]Md Faizul Bari, Shihabur Rahman Chowdhury, Reaz Ahmed, and Raouf Boutaba. „On orchestrating Virtual Network Functions“. In: *CNSM*. 2015, pp. 50–56 (cit. on p. 76).
- [14]John E Beasley. „OR-Library: distributing test problems by electronic mail“. In: *Journal of the operational research society* 41.11 (1990), pp. 1069–1072 (cit. on p. 125).
- [15]Irwan Bello, Hieu Pham, Quoc V Le, Mohammad Norouzi, and Samy Bengio. „Neural Combinatorial Optimization with Reinforcement Learning“. In: *arXiv:1611.09940* (2016) (cit. on pp. 6, 13, 14, 19, 22, 24, 62, 63, 92).
- [16]Yoshua Bengio, Nicolas Boulanger-Lewandowski, and Razvan Pascanu. „Advances in optimizing recurrent networks“. In: *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*. IEEE. 2013, pp. 8624–8628 (cit. on p. 41).

- [17]Yoshua Bengio, Andrea Lodi, and Antoine Prouvost. „Machine Learning for Combinatorial Optimization: a Methodological Tour d’Horizon“. In: *arXiv preprint arXiv:1811.06128* (2018) (cit. on p. 21).
- [18]Albert S Berahas, Majid Jahani, and Martin Takáč. „Quasi-newton methods for deep learning: Forget the past, just sample“. In: *arXiv preprint arXiv:1901.09997* (2019) (cit. on p. 40).
- [19]Christopher Berner, Greg Brockman, Brooke Chan, et al. „Dota 2 with Large Scale Deep Reinforcement Learning“. In: *arXiv preprint arXiv:1912.06680* (2019) (cit. on p. 2).
- [20]Dimitri P Bertsekas. „Nonlinear Programming“. In: *Journal of the Operational Research Society* 48.3 (1997), pp. 334–334 (cit. on p. 36).
- [21]Shalabh Bhatnagar and K Lakshmanan. „An online actor–critic algorithm with function approximation for constrained markov decision processes“. In: *Journal of Optimization Theory and Applications* 153.3 (2012), pp. 688–708 (cit. on p. 36).
- [22]Pierre Bonami, Andrea Lodi, and Giulia Zarpellon. „Learning a classification of mixed-integer quadratic programming problems“. In: *International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research*. Springer. 2018, pp. 595–604 (cit. on p. 12).
- [23]Vivek S Borkar. „An actor-critic algorithm for constrained Markov decision processes“. In: *Systems & control letters* 54.3 (2005), pp. 207–213 (cit. on p. 35).
- [24]Greg Brockman, Vicki Cheung, Ludwig Pettersson, et al. „Openai gym“. In: *arXiv preprint arXiv:1606.01540* (2016) (cit. on p. 55).
- [25]Yuri Burda, Harri Edwards, Deepak Pathak, et al. „Large-scale study of curiosity-driven learning“. In: *arXiv preprint arXiv:1808.04355* (2018) (cit. on p. 51).
- [26]Yuri Burda, Harrison Edwards, Amos Storkey, and Oleg Klimov. „Exploration by random network distillation“. In: *arXiv preprint arXiv:1810.12894* (2018) (cit. on pp. 51, 99).
- [27]Imran Ali Chaudhry and Abid Ali Khan. „A research survey: review of flexible job shop scheduling techniques“. In: *International Transactions in Operational Research* 23.3 (2016), pp. 551–591 (cit. on p. 64).
- [28]Xinyun Chen and Yuandong Tian. „Learning to perform local rewriting for combinatorial optimization“. In: *Advances in Neural Information Processing Systems*. 2019, pp. 6278–6289 (cit. on pp. 11, 93).
- [29]Runwei Cheng, Mitsuo Gen, and Yasuhiro Tsujimura. „A tutorial survey of job-shop scheduling problems using genetic algorithms—I. Representation“. In: *Computers & industrial engineering* 30.4 (1996), pp. 983–997 (cit. on p. 66).
- [30]Rami Cohen, Liane Lewin-Eytan, Joseph Seffi Naor, and Danny Raz. „Near optimal placement of Virtual Network Functions“. In: *INFOCOM. IEEE*. 2015, pp. 1346–1354 (cit. on p. 76).
- [31]Hanjun Dai, Bo Dai, and Le Song. „Discriminative embeddings of latent variable models for structured data“. In: *International conference on machine learning*. 2016, pp. 2702–2711 (cit. on p. 92).
- [32]Zihang Dai, Zhilin Yang, Yiming Yang, et al. „Transformer-xl: Attentive language models beyond a fixed-length context“. In: *arXiv preprint arXiv:1901.02860* (2019) (cit. on p. 33).
- [33]Gal Dalal, Krishnamurthy Dvijotham, Matej Vecerik, et al. „Safe exploration in continuous action spaces“. In: *arXiv preprint arXiv:1801.08757* (2018) (cit. on p. 35).
- [34]Yann N Dauphin, Razvan Pascanu, Caglar Gulcehre, et al. „Identifying and attacking the saddle point problem in high-dimensional non-convex optimization“. In: *Advances in neural information processing systems*. 2014, pp. 2933–2941 (cit. on p. 46).
- [35]Michel Deudon, Pierre Cournut, Alexandre Lacoste, Yossiri Adulyasak, and Louis-Martin Rousseau. „Learning heuristics for the TSP by Policy Gradient“. In: *Inter. Conf. on the integration of CP, AI, and OP*. Springer. 2018, pp. 170–181 (cit. on pp. 6, 14, 19, 21, 24, 91).

- [36]Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, et al. *OpenAI Baselines*. <https://github.com/openai/baselines>. 2017 (cit. on p. 55).
- [37]Prafulla Dhariwal, Christopher Hesse, Oleg Klimov, et al. *Openai baselines*. 2017 (cit. on p. 125).
- [38]Yan Duan, Xi Chen, Rein Houthoofd, John Schulman, and Pieter Abbeel. „Benchmarking deep reinforcement learning for continuous control“. In: *International Conference on Machine Learning*. 2016, pp. 1329–1338 (cit. on pp. 29, 117).
- [39]Patrick Emami and Sanjay Ranka. „Learning permutations with sinkhorn policy gradient“. In: *arXiv preprint arXiv:1805.07010* (2018) (cit. on p. 21).
- [40]Javier Garcia and Fernando Fernández. „A comprehensive survey on safe reinforcement learning“. In: *Journal of Machine Learning Research* 16.1 (2015), pp. 1437–1480 (cit. on p. 34).
- [41]Michael R Garey, David S Johnson, and Ravi Sethi. „The complexity of flowshop and jobshop scheduling“. In: *Mathematics of operations research* 1.2 (1976), pp. 117–129 (cit. on pp. 3, 64).
- [42]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*. 2019, pp. 15554–15566 (cit. on p. 11).
- [43]Jason Gauci, Edoardo Conti, Yitao Liang, et al. „Horizon: Facebook’s open source applied reinforcement learning platform“. In: *arXiv preprint arXiv:1811.00260* (2018) (cit. on p. 2).
- [44]Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N Dauphin. „Convolutional sequence to sequence learning“. In: *arXiv preprint arXiv:1705.03122* (2017) (cit. on p. 32).
- [45]Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. „Learning to forget: Continual prediction with LSTM“. In: (1999) (cit. on p. 125).
- [46]Xavier Glorot and Yoshua Bengio. „Understanding the difficulty of training deep feedforward neural networks“. In: *Proceedings of the thirteenth international conference on artificial intelligence and statistics*. 2010, pp. 249–256 (cit. on pp. 84, 125).
- [47]Ivo Grondman, Lucian Busoniu, Gabriel AD Lopes, and Robert Babuska. „A survey of actor-critic reinforcement learning: Standard and natural policy gradients“. In: *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)* 42.6 (2012), pp. 1291–1307 (cit. on p. 61).
- [48]Abhishek Gupta, M Farhan Habib, Pulak Chowdhury, Massimo Tornatore, and Biswanath Mukherjee. „On service chaining using Virtual Network Functions in network-enabled cloud systems“. In: *ANTS*. 2015 (cit. on p. 76).
- [49]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 (cit. on p. 17).
- [50]Peter Henderson, Joshua Romoff, and Joelle Pineau. „Where did my optimum go?: An empirical analysis of gradient descent optimization in policy gradient methods“. In: *arXiv preprint arXiv:1810.02525* (2018) (cit. on p. 40).
- [51]Ashley Hill, Antonin Raffin, Maximilian Ernestus, et al. *Stable Baselines*. <https://github.com/hill-a/stable-baselines>. 2018 (cit. on p. 55).
- [52]Sepp Hochreiter. „Untersuchungen zu dynamischen neuronalen Netzen“. In: *Diploma, Technische Universität München* 91.1 (1991) (cit. on p. 17).
- [53]Sepp Hochreiter and Jürgen Schmidhuber. „Long short-term memory“. In: *Neural computation* 9.8 (1997), pp. 1735–1780 (cit. on p. 17).
- [54]John J Hopfield and David W Tank. „“Neural” computation of decisions in optimization problems“. In: *Biological cybernetics* 52.3 (1985), pp. 141–152 (cit. on p. 13).
- [55]IBM ILOG. „Cplex optimization studio“. In: *URL: http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer* (2014) (cit. on p. 84).

- [56]Max Jaderberg, Volodymyr Mnih, Wojciech Marian Czarnecki, et al. „Reinforcement learning with unsupervised auxiliary tasks“. In: *arXiv preprint arXiv:1611.05397* (2016) (cit. on p. 37).
- [57]Sham M Kakade. „A natural policy gradient“. In: *Advances in neural information processing systems*. 2002, pp. 1531–1538 (cit. on p. 54).
- [58]Richard M Karp. „Reducibility among combinatorial problems“. In: *Complexity of computer computations*. Springer, 1972, pp. 85–103 (cit. on p. 92).
- [59]Michael N Katehakis and Arthur F Veinott Jr. „The multi-armed bandit problem: decomposition and computation“. In: *Mathematics of Operations Research* 12.2 (1987), pp. 262–268 (cit. on p. 19).
- [60]Elias Khalil, Hanjun Dai, Yuyu Zhang, Bistra Dilkina, and Le Song. „Learning combinatorial optimization algorithms over graphs“. In: *Advances in Neural Information Processing Systems*. 2017, pp. 6348–6358 (cit. on pp. 14, 22, 26, 92).
- [61]Elias Boutros Khalil, Pierre Le Bodic, Le Song, George Nemhauser, and Bistra Dilkina. „Learning to branch in mixed integer programming“. In: *Thirtieth AAAI Conference on Artificial Intelligence*. 2016 (cit. on pp. 11, 22).
- [62]Sanghyeok Kim, Sungyoung Park, Youngjae Kim, Siri Kim, and Kwonyong Lee. „VNF-EQ: dynamic placement of Virtual Network Functions for energy efficiency and QoS guarantee in NFV“. In: *Cluster Comp.* 20.3 (2017), pp. 2107–2117 (cit. on p. 76).
- [63]Diederik P Kingma and Jimmy Ba. „Adam: A method for stochastic optimization“. In: *arXiv preprint arXiv:1412.6980* (2014) (cit. on pp. 41, 125).
- [64]Aldebaro Klautau, Nuria González-Prelcic, and Robert W Heath. „LIDAR data for deep learning-based mmWave beam-selection“. In: *IEEE Wireless Communications Letters* 8.3 (2019), pp. 909–912 (cit. on p. 93).
- [65]Wouter Kool, Herke Van Hoof, and Max Welling. „Attention, learn to solve routing problems!“ In: *arXiv preprint arXiv:1803.08475* (2018) (cit. on pp. 6, 14, 19, 21, 22, 24, 91).
- [66]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 (cit. on p. 12).
- [67]Alexandre Laterre, Yunguan Fu, Mohamed Khalil Jabri, et al. „Ranked reward: Enabling self-play reinforcement learning for combinatorial optimization“. In: *arXiv preprint arXiv:1807.01672* (2018) (cit. on p. 93).
- [68]Timothy P Lillicrap, Jonathan J Hunt, Alexander Pritzel, et al. „Continuous control with deep reinforcement learning“. In: *arXiv preprint arXiv:1509.02971* (2015) (cit. on p. 19).
- [69]Marcelo Caggiani Luizelli, Leonardo Richter Bays, Luciana Saete Buriol, Marinho Pilla Barcellos, and Luciano Paschoal Gaspary. „Piecing together the NFV provisioning puzzle: Efficient placement and chaining of Virtual Network Functions“. In: *Inter. Symp. on Integrated Net. Mgmt.* 2015, pp. 98–106 (cit. on p. 76).
- [70]Minh-Thang Luong, Hieu Pham, and Christopher D Manning. „Effective approaches to attention-based neural machine translation“. In: *arXiv preprint arXiv:1508.04025* (2015) (cit. on pp. 20, 31).
- [71]Brian Mahadevan. „Operations management“. In: *Theory And Practice* 5 (2010) (cit. on pp. 66, 124).
- [72]Hongzi Mao, Mohammad Alizadeh, Ishai Menache, and Srikanth Kandula. „Resource management with deep reinforcement learning“. In: *Proceedings of the 15th ACM Workshop on Hot Topics in Networks*. 2016, pp. 50–56 (cit. on pp. 14, 21).
- [73]Hongzi Mao, Malte Schwarzkopf, Shaileshh Bojja Venkatakrisnan, Zili Meng, and Mohammad Alizadeh. „Learning scheduling algorithms for data processing clusters“. In: *Proceedings of the ACM Special Interest Group on Data Communication*. 2019, pp. 270–288 (cit. on p. 14).

- [74]Antonio Marotta, Fabio D'Andreagiovanni, Andreas Kassler, and Enrica Zola. „On the energy cost of robustness for green Virtual Network Function placement in 5G virtualized infrastructures“. In: *Comp. Net.* 125 (2017), pp. 64–75 (cit. on pp. 76, 78).
- [75]Stephen McAleer, Forest Agostinelli, Alexander Shmakov, and Pierre Baldi. „Solving the Rubik's Cube Without Human Knowledge“. In: *arXiv preprint arXiv:1805.07470* (2018) (cit. on p. 93).
- [76]Rashid Mijumbi. „Placement and scheduling of functions in Network Function Virtualization“. In: *arXiv:1512.00217* (2015) (cit. on p. 76).
- [77]Rashid Mijumbi, Juan-Luis Gorricho, Joan Serrat, et al. „Design and evaluation of learning algorithms for dynamic resource management in Virtual Networks“. In: *NOMS*. 2014, pp. 1–9 (cit. on p. 77).
- [78]Rashid Mijumbi, Juan-Luis Gorricho, Joan Serrat, et al. „Neural Network-based autonomous allocation of resources in Virtual Networks“. In: *EuCNC*. 2014, pp. 1–6 (cit. on p. 77).
- [79]Marvin Minsky. „Steps toward artificial intelligence“. In: *Proceedings of the IRE* 49.1 (1961), pp. 8–30 (cit. on p. 114).
- [80]Azalia Mirhoseini, Hieu Pham, Quoc V Le, et al. „Device placement optimization with Reinforcement Learning“. In: *ICML, Volume 70*. 2017, pp. 2430–2439 (cit. on pp. 15, 19, 23, 52).
- [81]Volodymyr Mnih, Adria Puigdomenech Badia, Mehdi Mirza, et al. „Asynchronous methods for deep reinforcement learning“. In: *International conference on machine learning*. 2016, pp. 1928–1937 (cit. on pp. 28, 29).
- [82]Hendrik Moens and Filip De Turck. „VNF-P: A model for efficient placement of Virtualized Network Functions“. In: *CNSM*. 2014, pp. 418–423 (cit. on p. 76).
- [83]Mohammadreza Nazari, Afshin Oroojlooy, Lawrence Snyder, and Martin Takác. „Reinforcement Learning for Solving the Vehicle Routing Problem“. In: *Advances in NIPS*. 2018, pp. 9839–9849 (cit. on pp. 6, 14, 21, 24, 28, 34).
- [84]Yurii E Nesterov. „A method for solving the convex programming problem with convergence rate $O(1/k^2)$ “. In: *Dokl. akad. nauk Sssr*. Vol. 269. 1983, pp. 543–547 (cit. on p. 40).
- [85]Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. „On the difficulty of training recurrent neural networks“. In: *International conference on machine learning*. 2013, pp. 1310–1318 (cit. on p. 18).
- [86]Adam Paszke, Sam Gross, Soumith Chintala, et al. „Automatic differentiation in pytorch“. In: (2017) (cit. on pp. 44, 60, 125).
- [87]Santiago Paternain, Luiz Chamon, Miguel Calvo-Fullana, and Alejandro Ribeiro. „Constrained Reinforcement Learning Has Zero Duality Gap“. In: *Advances in Neural Information Processing Systems*. 2019, pp. 7553–7563 (cit. on p. 35).
- [88]Laurent Perron and Vincent Furnon. *OR-Tools*. Version 7.2. Google, July 19, 2019 (cit. on p. 66).
- [89]Jan Peters and Stefan Schaal. „Natural actor-critic“. In: *Neurocomputing* 71.7-9 (2008), pp. 1180–1190 (cit. on p. 54).
- [90]Tu-Hoa Pham, Giovanni De Magistris, and Ryuki Tachibana. „Optlayer-practical constrained optimization for deep reinforcement learning in the real world“. In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE. 2018, pp. 6236–6243 (cit. on p. 35).
- [91]Ning Qian. „On the momentum term in gradient descent learning algorithms“. In: *Neural networks* 12.1 (1999), pp. 145–151 (cit. on p. 40).
- [92]Roberto Riggio, Abbas Bradai, Tinku Rasheed, et al. „Virtual Network Functions orchestration in wireless networks“. In: *CNSM*. 2015, pp. 108–116 (cit. on p. 76).
- [93]Herbert Robbins and Sutton Monro. „A stochastic approximation method“. In: *The annals of mathematical statistics* (1951), pp. 400–407 (cit. on p. 39).

- [94]Krzysztof Rusek, José Suárez-Varela, Albert Mestres, Pere Barlet-Ros, and Albert Cabellos-Aparicio. „Unveiling the potential of Graph Neural Networks for network modeling and optimization in SDN“. In: *Proceedings of the 2019 ACM Symposium on SDN Research*. 2019, pp. 140–151 (cit. on p. 93).
- [95]Tim Salimans, Jonathan Ho, Xi Chen, Szymon Sidor, and Ilya Sutskever. „Evolution strategies as a scalable alternative to reinforcement learning“. In: *arXiv preprint arXiv:1703.03864* (2017) (cit. on p. 39).
- [96]Franco Scarselli, Marco Gori, Ah Chung Tsoi, Markus Hagenbuchner, and Gabriele Monfardini. „The graph neural network model“. In: *IEEE Transactions on Neural Networks* 20.1 (2008), pp. 61–80 (cit. on p. 92).
- [97]Tom Schaul, Diana Borsa, Joseph Modayil, and Razvan Pascanu. „Ray interference: a source of plateaus in deep reinforcement learning“. In: *arXiv preprint arXiv:1904.11455* (2019) (cit. on p. 36).
- [98]John Schulman, Sergey Levine, Pieter Abbeel, Michael Jordan, and Philipp Moritz. „Trust region policy optimization“. In: *International conference on machine learning*. 2015, pp. 1889–1897 (cit. on p. 53).
- [99]John Schulman, Philipp Moritz, Sergey Levine, Michael Jordan, and Pieter Abbeel. „High-dimensional continuous control using generalized advantage estimation“. In: *arXiv preprint arXiv:1506.02438* (2015) (cit. on p. 123).
- [100]John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. „Proximal policy optimization algorithms“. In: *arXiv preprint arXiv:1707.06347* (2017) (cit. on pp. 2, 54, 55).
- [101]Mark Shifrin, Erez Biton, and Omer Gurewitz. „Optimal control of VNF deployment and scheduling“. In: *2016 IEEE International Conference on the Science of Electrical Engineering (ICSEE)*. IEEE. 2016, pp. 1–5 (cit. on p. 76).
- [102]Nils T Siebel and Gerald Sommer. „Evolutionary reinforcement learning of artificial neural networks“. In: *International Journal of Hybrid Intelligent Systems* 4.3 (2007), pp. 171–183 (cit. on p. 35).
- [103]David Silver. *Lecture notes in Reinforcement Learning*. 2015 (cit. on p. 2).
- [104]David Silver, Thomas Hubert, Julian Schrittwieser, et al. „A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play“. In: *Science* 362.6419 (2018), pp. 1140–1144 (cit. on p. 93).
- [105]David Silver, Thomas Hubert, Julian Schrittwieser, et al. „Mastering chess and shogi by self-play with a general reinforcement learning algorithm“. In: *arXiv preprint arXiv:1712.01815* (2017) (cit. on pp. 2, 92).
- [106]Felipe Petroski Such, Vashisht Madhavan, Edoardo Conti, et al. „Deep neuroevolution: Genetic algorithms are a competitive alternative for training deep neural networks for reinforcement learning“. In: *arXiv preprint arXiv:1712.06567* (2017) (cit. on p. 117).
- [107]Ilya Sutskever, Oriol Vinyals, and Quoc V Le. „Sequence to sequence Learning with Neural Networks“. In: *Advances in NIPS*. 2014, pp. 3104–3112 (cit. on pp. 18, 30).
- [108]Richard S Sutton, Andrew G Barto, et al. *Introduction to Reinforcement Learning*. Vol. 135. MIT press Cambridge, 1998 (cit. on pp. 1, 14, 25, 111).
- [109]István Szita and András Lörincz. „Learning Tetris using the noisy cross-entropy method“. In: *Neural computation* 18.12 (2006), pp. 2936–2941 (cit. on p. 117).
- [110]Eric Taillard. „Benchmarks for basic scheduling problems“. In: *European journal of operational research* 64.2 (1993), pp. 278–285 (cit. on p. 125).
- [111]Aviv Tamar and Shie Mannor. „Variance adjusted actor critic algorithms“. In: *arXiv preprint arXiv:1310.3697* (2013) (cit. on p. 35).
- [112]Haoran Tang, Rein Houthoofd, Davis Foote, et al. „# exploration: A study of count-based exploration for deep reinforcement learning“. In: *Advances in neural information processing systems*. 2017, pp. 2753–2762 (cit. on p. 50).

- [113]Chen Tessler, Daniel J Mankowitz, and Shie Mannor. „Reward Constrained Policy Optimization“. In: *arXiv:1805.11074* (2018) (cit. on pp. 35, 36).
- [114]Yuandong Tian, Jerry Ma, Qucheng Gong, et al. „Elf opengo: An analysis and open reimplementaion of alphazero“. In: *arXiv preprint arXiv:1902.04522* (2019) (cit. on p. 99).
- [115]Kristof Van Moffaert and Ann Nowé. „Multi-objective reinforcement learning using sets of pareto dominating policies“. In: *The Journal of Machine Learning Research* 15.1 (2014), pp. 3483–3512 (cit. on p. 36).
- [116]Ashish Vaswani, Noam Shazeer, Niki Parmar, et al. „Attention is all you need“. In: *Advances in neural information processing systems*. 2017, pp. 5998–6008 (cit. on pp. 14, 16, 20, 66, 91).
- [117]Oriol Vinyals, Igor Babuschkin, Wojciech M Czarnecki, et al. „Grandmaster level in StarCraft II using multi-agent reinforcement learning“. In: *Nature* 575.7782 (2019), pp. 350–354 (cit. on p. 30).
- [118]Oriol Vinyals, Samy Bengio, and Manjunath Kudlur. „Order matters: Sequence to sequence for sets“. In: *arXiv preprint arXiv:1511.06391* (2015) (cit. on pp. 33, 34, 91).
- [119]Oriol Vinyals, Timo Ewalds, Sergey Bartunov, et al. „Starcraft ii: A new challenge for reinforcement learning“. In: *arXiv preprint arXiv:1708.04782* (2017) (cit. on p. 2).
- [120]Vinyals, Oriol and Fortunato, Meire and Jaitly, Navdeep. „Pointer Networks“. In: *Advances in NIPS*. 2015, pp. 2692–2700 (cit. on pp. 11–13, 16, 21, 22, 34, 92).
- [121]Network Functions Virtualisation. „An introduction, benefits, enablers, challenges & call for action“. In: *White Paper, SDN and OpenFlow World Congress*. 2012 (cit. on p. 73).
- [122]Ronald J Williams. „Simple statistical gradient-following algorithms for connectionist Reinforcement Learning“. In: *Machine learning* 8.3-4 (1992), pp. 229–256 (cit. on pp. 36, 119).
- [124]Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. „Aggregated residual transformations for deep neural networks“. In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2017, pp. 1492–1500 (cit. on p. 20).
- [125]Yuehua Xu, David Stern, and Horst Samulowitz. „Learning adaptation to solve constraint satisfaction problems“. In: *Proceedings of Learning and Intelligent Optimization (LION)* (2009) (cit. on p. 14).
- [126]Haipeng Yao, Bo Zhang, Peiying Zhang, et al. „RDAM: A Reinforcement Learning Based Dynamic Attribute Matrix Representation for Virtual Network Embedding“. In: *Trans. on Emerging Topics in Comp.* (2018) (cit. on p. 77).
- [127]Giulia Zarpellon, Jason Jo, Andrea Lodi, and Yoshua Bengio. „Parameterizing Branch-and-Bound Search Trees to Learn Branching Policies“. In: *arXiv preprint arXiv:2002.05120* (2020) (cit. on p. 12).
- [128]Jingzhao Zhang, Tianxing He, Suvrit Sra, and Ali Jadbabaie. „Why Gradient Clipping Accelerates Training: A Theoretical Justification for Adaptivity“. In: *International Conference on Learning Representations*. 2019 (cit. on p. 57).
- [129]Brian D Ziebart. „Modeling purposeful adaptive behavior with the principle of maximum causal entropy“. In: (2010) (cit. on p. 47).

Websites

- [123]Wurmen. *Genetic Algorithm for Job Shop Scheduling*. URL: <https://github.com/wurmen/Genetic-Algorithm-for-Job-Shop-Scheduling-and-NSGA-II> (cit. on p. 124).

Declaration

Herewith I acknowledge that I have completed this work solely and only with the help of the mentioned references.

Bilbao, October, 2020

Ruben Solozabal

