Network Aware Compute and Memory Allocation in Optically Composable Data Centres with Deep Reinforcement Learning and Graph Neural Networks

ZACHARAYA SHABKA[1,*] AND GEORGIOS ZERVAS[1]

[1] Optical Networks Group, Department of Electronic and Electrical Engineering, University College London, Roberts Building, Torrington Place, London, WC1E 7JE, United Kingdom
[*] Corresponding author: zacharaya.shabka.18@ucl.ac.uk

Compiled February 1, 2023

Composable data centre architectures promise a means of pooling resources remotely within data centres, allowing for both more flexibility and resource efficiency underlying the increasingly important infrastructure-as-a-service business. This can be accomplished by means of using an optically circuit switched backbone in the data centre network (DCN); providing the required bandwidth and latency guarantees to ensure reliable performance when applications are run across non-local resource pools. However, resource allocation in this scenario requires both server-level *and* network-level resource to be co-allocated to requests. The online nature and underlying combinatorial complexity of this problem, alongside the typical scale of DCN topologies, makes exact solutions impossible and heuristic based solutions sub-optimal or non-intuitive to design. We demonstrate that *deep reinforcement learning*, where the policy is modelled by a *graph neural network* can be used to learn effective *network-aware* and *topologically-scalable* allocation policies end-to-end. Compared to state-of-the-art heuristics for network-aware resource allocation, the method achieves up to 20% higher acceptance ratio; can achieve the same acceptance ratio as the best performing heuristic with $3\times$ less networking resources available and can maintain all-around performance when directly applied (with no further training) to DCN topologies with $10^2\times$ more servers than the topologies seen during training.

http://dx.doi.org/10.1364/ao.XX.XXXXXX

1. INTRODUCTION

Contemporary data centre network (DCN) architectures are based on (opto-)electronically packet-switched (EPS) networks. In a typical Cloud computing model, large tasks requiring more than one server's worth of resources for a long period of time can be distributed across numerous servers, which are all connected via this underlying EPS infrastructure. These allocations can not be done across single inter-server resource pools, instead requiring the task to be split into smaller tasks where each will be allocated resources from and run on a single server [1]. This limitations stems from two things primarily.

Firstly, bandwidth is limited by the bandwidth per-port of the opto-electronic switches which is fixed per-model. Popular EPS switches typically support a per-port bandwidth at the order of $\mathcal{O}(1Gb/s) - \mathcal{O}(10Gb/s)$, whereas intra-server communications (e.g. L1-cache access by the CPU) often operate at the order of $\mathcal{O}(Tb/s)$. For resources to access each other remotely, a large number of ports would be required for just a single pair of devices and networking component costs would increase. Such devices also have fixed bandwidth, meaning higher bandwidth servers require network infrastructure replacement or must be run sub-optimally.

Secondly, the unpredictable queuing patterns in packet-switched networks lead to non-deterministic latency. Compute mediums (i.e. CPU, GPU, RAM) co-located on the same server exchange information at very high rates. For example, L1 cache latency on high-end desktop CPUs exist in the $\mathcal{O}(ns) - \mathcal{O}(10ns)$ range. Application performance is strongly dependent on compute-memory latency [2]. EPS networks are incapable of consistently supporting standard application performance given typical forwarding latency is non-deterministic and in the range $\mathcal{O}(10\mu s) - \mathcal{O}(100\mu s)$.

These two features lead to *resource fragmentation* - where resources are available on a server but not accessible due to all the other resources on that server being occupied - and *inflexible resource-pooling* applications can run directly on a single pool who's size is limited by the amount of resources at a single server since inter-server pools are not possible.

Resource disaggregated DCs are a proposed DC architecture

supporting a network architecture that provides sufficient bandwidth and latency guarantees for resource pools to be defined across servers on which long-lived resource-hungry applications can be run with local-like performance. This would reduce fragmentation, as well as increase pooling flexibility. Such architectures can in fact be built using off-the-shelf commodity hardware such as commercially available optical-circuit switches [3–6] (note that these systems are not yet deployed commercially but have been shown beneficial in proof of concept demonstrations).

However, since both server- and network- resources need to be explicitly provisioned in order to allocate both compute and connectivity, allocation is more complex (it is a NP-hard combinatorial optimisation problem) as decisions need to be made across the product of both of these domains, rather than only server resources as in conventional resource management frameworks [7–10]. This paper will refer to this requirement as *network aware resource allocation*. On graphs with the order of $\mathbf{O}(10^2)$ nodes - where DCN topologies are at least this large - exact solutions are computationally intractable, as well as innapropriate for online allocation scenarios where requests arrive dynamically rather than all at once. Moreover, while heuristic solutions can generate acceptable outcomes in reasonable time, they are sub-optimal and in particular subject to designer bias which can considerably limit performance and/or scalability [4, 5, 11, 12].

This paper shows that deep reinforcement learning (DRL) with graph neural network (GNN) based policies can learn very effective network aware allocation policies end-to-end which both outperform and out-scale conventional methods. Acceptance rate, CPU utilisation and memory utilisation are improved by up to 19%, 24% and 22% respectively compared to state-of-the-art heuristics. Furthermore the DRL-based method achieves approximately the same performance as the best heuristic achieves whilst requiring $3\times$ less network resources to do so. While the method is trained on small DCN topologies with $\mathcal{O}(10^1)$ servers, the GNN-based policy architecture is topology-size agnostic. Because of this it can be directly applied to topologies with $10^2\times$ more servers than seen during training and maintain it's allocation performance without further training required. Following this, a discussion on interpreting the learnt policy is presented, indicating that the method is flexible under changing network resource profiles and generally more adaptable than the heuristics.

2. BACKGROUND

A. Deep Reinforcement Learning

Reinforcement learning (RL) relates to the study of how to behave optimally in dynamic environments. An environment is formally described by a *Markov Decision Process* (MDP) defined as a tuple $< S, A, R_a, T_a >$ where: S is the set of all possible states that the MDP can be in, A is the set of all possible actions that some actor can take in this environment, R_a is a function describing the reward yielded when an agent is in state s, takes action a and ends up in state s', and T_a is a function describing the probability of an agent being in state s, taking action a and ending up in state s'. A policy is described as some function, $\pi(s) \rightarrow a$ which maps states to actions, where RL problems seek to find the policy that will yield the largest reward over time. Deep reinforcement learning (DRL) refers to when policy functions are modelled using deep neural networks. An extensive explanation of RL theory can be found in [13]. This study has yielded impressive performance on complex tasks such as the board game Go or the video game Starcraft [14, 15].

A.1. Graph Neural Networks

Graph neural networks (GNN) extend standard neural network (NN) architectures to graph-structured data, where data points are nodes and any relationships. GNNs attempt to account for topological information as well as the raw data during learning tasks by means of a *message passing procedure* where node and/or edge information is propagated through the network via single-hop neighbour exchanges and aggregated using a layer of the GNN. This procedure can be represented by the equation: $h_v = g(v, \sum_{v' in N(v)} f(v', e_{v,v'}))$, where g and f are (learnable/NN-based) functions, v is the information at node v, $e_{v,v'}$ is the information at the edge connecting nodes v and v', $N(v)$ is the set of all one-hop neighbours of node v and h_v is the new representation of node v after a single message passing procedure has been applied. GNN architectures have been shown to outperform classical graph-embedding methods in statistical tasks such as graph clustering or node classification. GraphSAGE is a particular GNN architecture which introduced a design where the NN-model is agnostic to the shape of the dataset's topology alongside a more efficient neighbour-sampling technique used during message passing. This allows for more generalised and scalable graph-based learning and many contemporary GNN architectures are based on these design choices [16–18].

A.2. Combinatorial Optimisation

Combinatorial optimisation (CO) problems describe scenarios with a finite set of items, where some optimal sub and/or ordered set of items (the *solution*) must be determined. Formally, a CO problem can be described by a set of problem instances, I; for some instance $x \in I$, $f(x)$ is the set of valid solutions and $m(x, y)$ (termed the objective function) maps some valid solution $y \in f(x)$ to some number; the goal of a CO problem is to find, for some $x \in I$, a solution $y' \in f(x)$ such that $m(x, y')$ is either minimised or maximised (depending on the problem).

A.3. RL and GNNs for CO

Graph-based combinatorial optimisation problems were presented as DRL problems initially in [19]. A node-by-node solution framework with DRL policies modelled with GNNs was proposed where GNNs generate node-embeddings which are used to determine whether that node should be added to the solution set. Further iterations of this framework have shown it able to scale to large graphs [20–22] and perform competitively against exact solvers [23]. This architecture has been used to effetively solve a number of network and computer-system based optimisation problems, such as distributed machine learning [24], cluster management with dependency-structured tasks [25], optical routing [26] and virtual network embedding [27, 28]. However, there has never been an application of this method to the kind of system (optically composable data centres) and allocation problem (network aware resource allocation) presented here, and as such these works are not directly comparable.

A.4. (Composable) Data Centre Resource Allocation

Network aware algorithms for optically composable disaggregated data centres are described in [4, 5, 11]. These works augment a breadth-first-search procedure to recursively discover sub-networks that can support the required compute, memory, storage, bandwidth and/or latency by some given request. They

show advantages over traditional packing algorithms such as best fit, but suffer from poor scalability since they are exhaustive in the worst case. A bandwidth-aware multi-resource cluster allocation (and scheduling) method is described in [12], where servers are ranked based on server-local compute and network resources. This method has more limited exposure to the network as it does not consider network resources multiple hops away from the server, but is less exhaustive in it's search complexity (elaborated in section B.1).

Cluster scheduling in commercial DCs is often handled using a variant of the dominant resource fairness algorithm and typically accounts for numerous practical features relating to operational efficiency such as software package availability of each machine or failure domain acknowledgement, and generally seeks to reduce fragmentation and increase utilisation [7, 9, 10, 29]. The framework presented in [30] describes a generalised resource allocation that implements a highly paralellised GPU-based packing method to solve generic virtual machine allocation in a typical (not composable/optical) DC environment. Similarly, the virtual data centre (VDC) mapping problem is implemented using a heuristic based on hand-crafted objectives for server, switch and link mapping [31] - though only comparisons against random allocation are made. While not directly comparable to the method presented here which addresses a fundamentally different type of computer system and associated problem, it is of value to note these methods as they provide context for the motivations of composable DCs, as well as the difficulties of tackling NP-hard allocation problems in such systems.

3. PROBLEM

We identify resource allocation in composable DCs as a CO problem, represent it as a MDP and show that GNN-based DRL solutions can learn superior allocation strategies compared to several baselines. Previous work has shown that DRL with GNN-based policies can be used to learn network aware resource allocation algorithms in composable data centres which are both high performing and scalable [32]. This work continues the examination of these architectures in a similar experiemental setting, but provides more extensive analysis of the nature of the policy that is learnt. Specifically, our analysis is extended to the network usage, fairness and general nature of the decision making by the DRL agent. In this way we seek to understand more intuitively what the agent is doing, as opposed to a more simple observation of improved long-term allocation outcomes. We discuss new results about how each tier's network resources are used by the agent (and comparative methods) across the test topologies, as well as analyse the relationship between request size and how the agent's allocations are distributed within racks, between racks and between clusters.

A. Defining the Markov Decision Process

Environment: The environment consists of a DCN (a set of servers and a network interconnecting these servers via some network switches) and requests which arrive one by one. We model a 3-tier space-switched DCN architecture (standard in modern data centres and in-line with the previous optically composable DC architectures detailed in past literature). Servers connect directly to a tier-1 switch, which itself connects to a number of tier-2 switches, which in turn connect to a number of tier-3 switches. A set of servers connected to the same tier-1 switch are referred to as a *rack*. A set of racks who's tier-2 switches connect to the same set of tier-2 switches is referred to as a *cluster*. By varying the number of channels-per-link at each tier, we generate 9 distinct topologies. These topologies span 3 different oversubcription ratios and each also have different absolute quantities of network resources. Table 1 shows explicitly how these topologies were generated and Fig. 1 visualises the DRL learning loop with this MDP.

The DCN is represented by a graph, $G(V, E)$. Each server (represented by the set of nodes $V \in G$) has an associated resource vector. Specifically, in this problem the CPU and memory resources are accounted for so that $[v_{cpu}, v_{mem}] \forall v \in V$, where v_{cpu} & v_{mem} represent the available CPU and memory resources of server v at any given moment. Each node is initialised at the start of an episode as $v = [16, 16]$ meaning there are 16 discrete units available of CPU and memory resources at each server initially.

Similarly, each switch has a particular number of input and output ports. This is represented by proxy using edge features, such that each edge has a number of distinct channels, also denoted by a resource vector $[e_{ch}] \forall e \in E$ where e_{ch} is the number of available channels in that link. This represents a scenario where a certain number of ports on a switch are reserved for a particular server who's direct link to that switch has a certain number of channels.

Table 1. Oversubscription and number of channels per link for each topology. 'Bottom-top oversubscription' refers to the oversubscription from the servers to the top tier of switches (tier-3). 'Oversub' refers to the oversubscription at the interface between that tier and the tier below it (hence Tier-1 does not have an 'oversub' value. In this work we used topologies of this structure with $n \in \{8, 16, 32\}$.

Oversubscription, channels-per-link for network tiers			
	Bottom-top Oversubscription		
	1:16	1:8	1:4
	Ovsub, Chan	Ovsub, Chan	Ovsub, Chan
Tier-1	-, n	-, n	-, n
Tier-2	1:4, $2n$	1:2, $4n$	1:2, $4n$
Tier-3	1:4, $\frac{1}{2}n$	1:4, n	1:2, $2n$

Requests arrive at the DCN one at a time and are not seen in advance, where $R = [r_{cpu}, r_{mem}, r_t]$ represents the CPU, memory and holding time requirements for that request. The DCN environment is event-driven such that a ticker iterates once everytime a new request is received. As such when a request is allocated, its holding time refers to after how many future request arrivals it will de-allocate. A valid solution consists of a set of nodes, $V^{'}$, must be found to satisfy the constraints of 1. $\sum v_x \in V^{'} \geq r_x, x \in \{cpu, mem\}$ and 2. a set of $\frac{|V^{'}|(|V^{'}|-1)}{2}$ distinct paths can be found to guarantee all-to-all connectivity between all $v \in V^{'}$. A good solution to this problem is one which maximises successful allocations over time.

Episode: An episode is defined as receiving N requests. For each request the agent will iteratively choose servers until either it has successfully allocated a request or it's solution is invalid. When a new request arrives, if there is enough raw resources

available in the DCN to theoretically allocate it, the agent is prompted to attempt to do so. If an invalid solution is generated (there are not enough network resources available to connect all servers in the allocation) then this request is dropped and a new request is fetched. We do not model queuing as this involves another algorithmic domain (to balance new and queued request with priority structures etc) and in this work we wish to focus on the network-awareness/congestion dynamics of the underlying system. Moreover, queuing/prioritising can be handled in parallel to resource allocation decisions so can be considered separately.

State: given an awaiting request $R = \{r_{cpu}, r_{mem}, r_t\}$ (CPU, memory and holding-time requirements respectively) the MDP's state is $s = \{G(V_{norm}, E), r_t, U_{cpu}, U_{mem}\}$ where, $V_{norm} = \{\frac{v_x}{r_x} \forall v_x \in V\}, x \in \{cpu, mem\}$ and U_x is the DCN-global utilisation of resource $x \in \{cpu, mem\}$. This combines both node-, edge- and graph- level resource information within the state representation.

Feature normalisation is important in machine learning problems to ensure that certain features with larger absolute values/variation do not disproportionately influence policy updates during training. Furthermore, normalised state representation ensures that policies should be robust under testing on similar environments differing only by absolute scales (e.g. a DCN with 4 CPU units-per-server vs one with 16 units-per-server, or a scenario where requests are between 1-8 servers vs one where they are between 8-16). As such, each server's resource values are normalised with respect to the current requested amount of that resource. Similarly, link-resources are normalised with respect to the maximum initial amount on any link in the DCN. This ensures that the policy is exposed to the DCN in a way that is feature-scale agnostic, as well as agnostic to the relative scale of request quantities and server-resource quantities.

Action: A server $v \in V$ is added to V'. If constraint 2 defined previously cannot be satisfied (when k=3 shortest paths are tried per node-pair) the allocation fails. If constraint 2 is satisfied, then $max(v_x, r_x)$ - the maximum that can be provisioned from this server up to the limit of how much is needed and how much is present at that server - is allocated from v ($x \in \{cpu, mem\}$) and one channel-per-link per-path per-node-pair are allocated. If constraint 1 & 2 are satisfied the allocation is successful. Per request, actions are taken until an action succeeds or fails an allocation, at which point a new request is fetched (until termination).

Reward: The reward should guide the policy to maximise the number of requests allocated over time. The reward should also not be obviously biased towards certain requests (e.g. larger vs smaller ones) and not be complex in order to avoid excessive hand-crafting and to give the policy freedom to find allocation strategies not limited by designer bias. We craft a reward which awards α, β and γ for server selections resulting in successful allocations, unsucessful allocations and neither (allocation has not failed but is not fully provisioned yet) respectively.

We set $\gamma = 0$ since otherwise rewards are biased in favour of allocations with more or less servers involved when $\gamma > 0$ and $\gamma < 0$ respectively. We also set $\alpha = -\beta = 10$ - found by simple grid search - to ensure that the reward creates stronger signals contrasting good (lots of allocations) and bad (lots of failures) outcomes compared to if only successful requests were rewarded, or if only failed requests were punished. Further tuning might determine that performance improves with unequal magnitudes of rewards for successes and failures, thought we leave this for future work due to the open-endedness of the exercise. Furthermore, more complex rewards can often lead to unexpected/unwanted behaviour. For example, rewarding acceptance more than failures are punished may lead to prioritising acceptance at all costs, neglecting larger requests that are harder to allocate and prioritising easier smaller ones exclusively.

While CPU and memory utilisation are observed in testing, we do not account for them explicitly in the reward. This is largely because utilisation is inherently request-dependent and accounting for it in the reward requires greater consideration and hand-crafting than using only acceptance. Consider the case where the reward is proportional to utilisation. Utilisation increases more when a larger request is allocated. Some non-intuitive considerations would therefore have to be made to ensure fair rewards are given to successful allocations across different request sizes (i.e. not systematically prioritising larger requests by giving greater rewards for allocating them). Furthermore, since higher acceptance means more requests allocated in the DCN at any given time, better acceptance rates (as is seen in section 5) leads to higher utilisation even without explicit handling in the reward structure. As such, we use acceptance only in order to reduce reward complexity. We nevertheless believe that exploring different reward structures is an interesting avenue of future work.

B. Defining the deep reinforcement learning model

The learning model consists of a GNN (based on the GraphSAGE architecture [17]) and 2 deep neural networks (DNN) which we refer to as DNN_1 and DNN_2. The GNN acts on $G(V_{norm}, E)$ to generate embeddings of each node in the topology, where each layer of the GNN was constructed with 1 hidden layer of 16 units. DNN_1 outputs a high dimensional (8 units) representation of $[r_t, U_{cpu}, U_{mem}]$. DNN_2 then calculates logits for each node in the DCN graph based on an input of the concatenation of the GNNs embedding of that node, the output of DNN_1 and the element-wise mean of the embeddings of the nodes that have already been allocated to that request (or a zero-vector if the request has just been received and nothing has been allocated yet). These logits are passed through a softmax function to specify the probability of choosing each node. All hidden layers use ReLU activation. This model is used to approximate a policy, trained using the proximal policy optimisation (PPO) RL algorithm, and is implemented using RLlib and Deep Graph Library [33, 34].

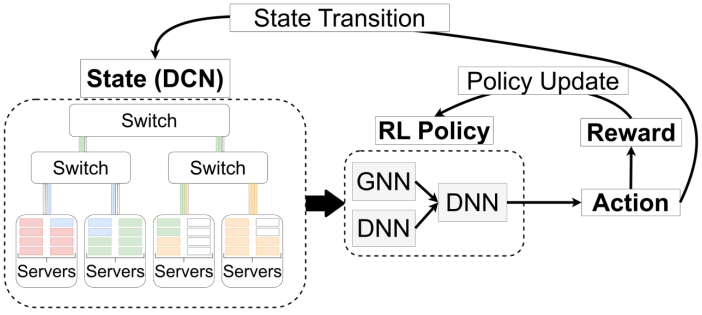

Fig. 1. Visualisation of the DRL feedback loop. The environment state is the DCN. Different coloured servers/links refer to their allocation to different requests (distinguished by colour).

The GNN used a distinct mean-based aggregator for each layer, where messages exchanged during the message passing

process are aggregated like: $v = \frac{1}{|N(v)|+1} \sum_{x \in M_v} W_i(x)$ where v is a node (embedding) in the DCN graph $G(V, E)$, $N(v)$ is the set of the one-hop neighbours of v, M_v is the set of messages received by v from it's neighbours and W_i is a neural network associated with the i^{th} layer of the GNN.

The GNN outputs embeddings of each node in 16 dimensions, and 3 layers were used so that information from the top of the network can be accounted for in the embeddings of the servers.

We compare our model against 3 baselines (Tetris, NALB, NULB) from previous work as well as a random allocation policy [5, 11, 12]. By training and testing the same DRL model across each of these topologies using the same reward structure we learn a distinct policy per topology and seek to explore if the model is topologically flexible in the policy it can learn given the underlying environment.

4. EXPERIMENTAL SETUP

A. Training and Testing

Training is implemented separately on each topology. Topologies used for training have 64 nodes (2 clusters $\times$ 2 racks $\times$ 16 servers-per-rack) and servers have 16 units of CPU and memory resources each. Requests are uniformly sampled with a maximum request size of 8 full-servers worth of resources in each domain, and their holding times are sampled such that the average offered load on the DCN system is 95% of all CPU and memory resources. Separate agents are trained for each topology, and tested against each baseline on that same topology. Training episodes are terminated after 32 requests have been received (successfully or otherwise) by the agent, and testing episodes on the smaller topology are terminated after 128 requests have been received. Agent's are tested on the same topologies on which they were trained. On a Nvidia A100 GPU and a Intel Xeon Gold 6148 CPU, training converges within $\approx$ 4 hours over $\approx$ 1000 episodes.

Shorter training episodes are generally more desirable since long episodes can create difficult learning scenarios [35, 36]. On the other hand, in the scenario explored here, any trained agent deployed in a real DC would be operating in a continuous environment. As such it is important that the policy can be trained on episodes that are short, but not too short such that the long-term dynamics of the system are unobserved in training, and thus not accounted for in deployment. Our training cluster is small (64 nodes, 4 racks of 16 servers) but still has servers divided into racks and clusters such that the different tiers of the network must be accounted for in allocations. We choose for requests to have a maximum size of $\frac{1}{2}$ a rack (based approximately on [1]) and be uniformly distributed. We uniformly distribute holding times within a range such that on average throughout the lifetime of an episode the offered load is 95% (after the 'warm up' period when utilisation starts at 0 and gradually climbs to some convergent value as requests begin to arrive in the system). As such there should be no requests arriving after warm-up which are trivial to allocate due to excessive amounts of free resources. This creates a challenging allocation environment in which network usage must be carefully accounted for due to the high demand on the system, and where on average $\frac{1}{2}$ of the training episode is spent in a high utilisation state (post warm-up).

Testing consists of deploying a trained policy on the same topology on which it was trained. Testing episodes (128 requests) are longer than training ones (32 requests) to emphasise performance on the long-term dynamics or stability of the system (more akin to real DC system operation). In this case, the episode spends approximately $\frac{7}{8}$ of the time in the high utilisation state so that the long-term stability of the method is observed. Testing episodes are seeded in the same way for each baseline, so all methods are exposed to the same set of requests (per test) received in the same order. Each test is run 5 times and results presented (where a single value per topology is shown) are the average of these 5 runs. Note that since testing/inference requires only a forward pass through the GNN-based policy network multiple times (once per each server selection until success or failure), and as such takes $\mathbf{O}(seconds)$ to select servers for a single request.

Tests are also implemented on scaled up ($\mathcal{O}(10^3)$ nodes) versions of each respective topology. Much longer episodes are required here since warm up takes much longer (since there is $\mathcal{O}(10^2)\times$ more resources in the DCN). Additionally, in order to maintain a high offered load the holding times are increased appropriately so that resource requirements build up in the system and allocation becomes harder as more requests arrive. More crucially, this test is done to explore the suitability of this method in a real DCN allocation scenario. Server clusters in large enterprise computer networks are of the order of $\mathcal{O}(10^3)$ servers and above. As such scalability to topologies of at least this scale is necessary.

If such a method were deployed in a real DC (though not the case in this paper which is entirely simulation based), some training and scaling practicalities must be considered. A small test cluster may be feasibly reserved for experimentation [37], though full-scale experimentation is not possible as this would require halting all or much of the services provided by the cluster (since the pre-trained allocation agent would be unsuitable for service level requirements). Where a sufficiently accurate simulation of DCN patterns is not available - which it often isn't [38–40] - and a small cluster is reserved for experimentation, any algorithm developed on the small cluster must be consistent with respect to performance when transferred to the larger one. For this reason we observe whether a policy trained on a topology with $\mathbf{O}(10^1)$ nodes can be directly deployed with nor modification or further training on a larger topology with $\mathbf{O}(10^3)$ servers (1024 specifically; 4 clusters of 4 racks each with 16 servers). Moreover, this also examines the longer-term stability of the method as the large-topology scenario - even more so than the longer testing episodes - requires allocation decisions to be made continuously for very long episodes ($\mathbf{O}(10^3)$ requests, 2048 specifically). This can therefore indicate if the policy is 'stable' (it doesn't deteriorate over time after a small number of episodes) or not (it can perform well on a small training episode, but does not do so consistently with larger topologies and/or longer episodes).

All methods are evaluated on the basis of three metrics; acceptance ratio, CPU utilisation and memory utilisation. Acceptance ratio refers to the proportion of all requests received by some allocation method that were successfully allocated. CPU and memory utilisation refers to the proportion of the total amount of that resource that is available in the DC which is currently allocated to some request. We also observe utilisation metrics relating to each tier of the network, as well as the characteristic relationship between request size and how distributed it is throughout the DC for a particular method. These (baseline or agent). These observations are considered as emergent features (rather than performance-based metrics used to evaluate allocation policies), and are used to try to understand the nature of what each method (the proposed RL-based one in particular)

does to achieve the allocation outcomes they do.

B. Baselines

Tetris [12] is a multi-resource packing heuristic. It uses the cosine similarity between task requirement and server resource availability vectors to calculate scores upon which packing decisions are made. Network resources are considered to be those which are present at a particular server (e.g. how many free communication channels are available at a particular node).Aa score penalty is imposed on non-local resources in order to encourage locality in its decision making. In this way an assumption about network resource efficiency is imposed which suggests that it is better to keep allocations rack local more than not.

NALB is a network-aware resource allocation algorithm which uses a bandwidth-weighted breadth-first-search algorithm and a bandwidth-weighted k-shortest paths routing algorithm to find suitable nodes and establish connectivity between them respectively [5]. The algorithm accounts for CPU, memory, storage (not used in this work), bandwidth and/or latency, where relative weighting between bandwidth and latency is a tunable parameter of the algorithm.

NULB works similarly to the NALB method, except that the breadth-first-search algorithm does not use weighting. Weighting is still used for the k-shortest paths procedure [5].

Random selects servers randomly. Used as a lower bound on expected performance.

B.1. Complexity comparison of DRL method and baselines

On a topology with set N nodes and set E edges, the complexity of server selection for the DRL method presented here and the Tetris baseline is $\mathbf{O}(|N|)$. Tetris requires a vector multiplication to be performed once per node, and message-passing - the core operation in the DRL-based method - has linear complexity with respect to number of nodes [41]. The NALB and NULB methods are breadth-first-search based, and as such have complexity $\mathbf{O}(|N| + |E|)$ since they are required to visit each and every node and edge in the exhaustive breadth-first-search procedure (in the worst case) before a viable server is guaranteed to be found. Random allocation has a trivial complexity of $\mathbf{O}(1)$.

As such it is seen that the Tetris and DRL-based method have the lowest complexity (excluding random allocation, which has trivially bad performance) and the NALB and NULB heuristics have the highest complexity.

5. RESULTS

A. DRL agent allocates more requests overall

On testing, the agent is observed to consistently outperform every baseline across each topology tested. For each topology, the percentage by which the agent improves over the best performing baseline on that topology is shown in Table 2. Most notably, it is seen that the agent thrives in particular when the network has few channels-per-links and/or when oversubscription is high (i.e. when the network is generally resource-constrained), achieving a 19.0%, 24.4% and 21.7% improvement for acceptance, CPU utilisation and memory utilisation respectively. Moreover, the agent is also able to find improvements even in the least resource-constrained environment where even the random baseline is comparable to some of the other baselines. It is also seen that, unsurprisingly, the agent achieves similarly improved resource utilisation for CPU and memory. In this case the same respective improvements are 5.8%, 2.7% and 2.7%. This is a natural emergent outcome of allocating more requests; higher acceptance ratio is equivalent to more requests occupying resources in the DC on average in a given moment in time. As such resource utilisation will also be higher on average.

Table 2. Percentage improvement of the agent pair over the second best performing baseline for that topology across all tested topologies.

RL agent improvement over best baseline (%) (acceptance, CPU util., memory util.)				
		Oversubscription		
		1:16	1:8	1:4
Chan.	8	19, 24, 22	15, 16, 20	22, 43, 23
per-link	16	9, 21, 16	11, 12, 15	8, 16, 11
tier-1	32	17, 17, 21	19, 12, 10	6, 3, 3

B. DRL agent is more consistent than baselines across different DCNs

The average performance at the end of the test episode for acceptance, CPU utilisation and memory utilisation are shown in Fig. 2. A key observation from these plots relating to both consistency and flexibility benefits is that while the RL method is always the best performing method on each topology, the baselines are frequently trading places for 2^{nd}, 3^{rd}, 4^{th} and 5^{th}.

As previously noted, heuristics are designed on the basis of some specific assumptions about a given system or problem, and are also tested in limited conditions. For example, the Tetris baseline assumes a statically defined emphasis on locality is beneficial, and also asserts that the network and node resources accounted for when scoring a particular server should be only it's local ones (i.e. it's directly attached resources, as opposed to accounting for resources from nodes/links up to k-hops away, for example). This is not so say that heuristics are entirely inflexible; Tetris parameterises how much of a penalty non-local servers should receive, and NALB parameterises the weighting between latency and bandwidth in the routing process for when both features are used. However, the fundamental decision making processes as well as what information is used to make these decisions are for the most part static after the design and testing phase. In this sense a heuristic has some inherent bias in it's behaviour that is derived from the design assumptions and test-performance feature-tuning, and as such do not necessarily have consistent performance benefits over some other heuristic in every circumstance. The tests who's results are shown in Table 2 and Fig. 2 differ only by the per-link resource quantity and share the fundamental topology. Even so, relatively simple variation is already enough to show the inconsistency of heuristics in this regard. Conversely, the agent learns appropriate policies for each network-resource profile and is able to consistently find better performance across each topology.

C. DRL agent is more consistent than baselines with respect to request size

The reward structure is designed to be minimally imposing on the kind of policies the agent can learn, and as such is related only to how many requests it successfully allocates rather than some request-specific information (e.g. the resource requirement

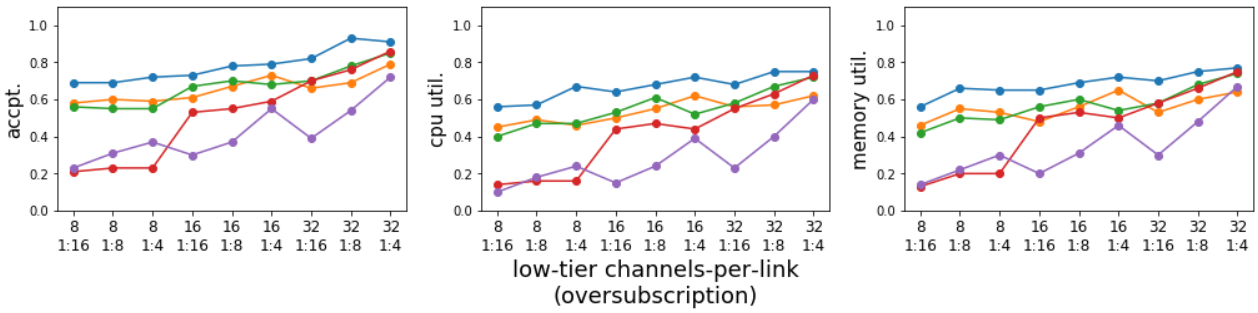

Fig. 2. blue=RL agent, orange=Tetris, green=NALB, red=NULB, purple=random. Line plots showing the acceptance ratio (left), CPU utilisation (middle) and memory utilisation (right) for each method when tested on each topology. Topology labels (x-axis) are channels per tier-1 link (upper part) and oversubcription (lower part).

magnitude). This was done to attempt to influence the agent to learn policies that are 'fair' with respect to any request it encounters. We expect from this design choice that the request should not learn to treat any particular size-range of request more carefully than others. As such we analyse how successful each agent was with requests of different sizes, grouped by the minimum possible number of servers required to allocate the request (this is a value between 1 and 8 inclusive since the largest requests ask for 8 full servers worth of resources). This is to see if the agent is better overall but systematically and significantly worse with requests of a particular size bracket, which would suggest a more biased policy is learnt.

What is seen is that the agent's advantage is very consistent across not just topologies but also request size brackets. The only exceptions are for the 3 most resource-constrained topologies, where the agent incurs a small deficit in the largest request size brackets. The worst case is seen on the 8-channel 1:8 oversubscription ratio topology, where deficit of 10 and 2 requests for the requests in the 7-server and 8-server size brackets respectively. This amounts to 1.8% of the total number of requests received across this test, compared to an overall improvement of 15%, where all other request-size brackets are improved relative to the baselines. The agent achieves better allocation outcomes not just overall but also with respect to each request size with few and minor exceptions.

D. The agent requires less networking resources for similar allocation performance

The agent's performance is also favourable compared against the baselines not just on the same topologies, but also across topologies. In particular, the agent can enable higher acceptance ratio on lower-resource topologies than the best performing baseline on higher-resource topologies. In the most extreme case observed across all experiments, the agent achieves an acceptance on the 8-channel 1:16 oversubscription topology that is only 1% lower than what the best performing baseline achieves on the 32-channel 1:16 oversubscription topology. In this case, the agent is effectively allowing for the same resource allocation service level to be achieved with 3× less resources.

Practically speaking, this is a very desirable feature. While optical DCN networks can allow various scalability issues to be avoided, the disaggregated resource paradigm that they enable imposes heavy demand on the network. An allocation policy that is able to minimise the amount of networking resources required to maintain some level of resource efficiency can allow for such systems to be built more feasibly. Large network infrastructure requirements (e.g. lots of fibre and switches) is expensive and requires much maintenance and planning [42]. Minimising these requirements is highly desirable to limit initial capital, operational and maintenance costs of such systems.

Table 3

RL agent performance delta on larger topologies (%)

(acceptance, CPU util., memory util.)

		Oversubscription		
		1:16	1:8	1:4
Low-tier	8.0	6, 4, 0	6, -12, -2	4, 13, 6
channels	16.0	16, 8, 14	10, 3, 10	9, 4, 11
per-link	32.0	7, 4, 11	-6, 8, 3	-1, -1, 6

E. Topology scale-up performance

Table 3 shows the percentage delta in test performance for the DRL agent when applied to each topology vs it's $\mathcal{O}(10^2)$ scaled up version. In the table $+ve$ indicates that the large-topology score was better and $-ve$ indicates that it was worse. Averaged across each topology, the delta is 5.6%, 3.4% and 6.5% for acceptance, CPU utilisation and memory utilisation respectively. The key indication from these results is simply that the agent is clearly able to learn a policy on a small graph that accounts for features of the topology and request distribution that are valid when applied to much larger topologies and over a much longer series of requests.

F. Interpreting the policy's allocation strategy

Here is presented a discussion, led by visual analysis, on the nature of the allocation policy that is learnt by the DRL agent. Since it is unclear how to directly extract policy principles from neural networks directly, numerical and visual analysis through experimental probing is required to infer as best as possible what the policy is doing. In effect, this section attempts to describe the learnt allocation policy as if it were a heuristic using several visualisations.

Fig. 3 shows the utilisation of each tier's network resources per-topology-per-method. The relationship between how many servers were allocated to a request and their distribution is

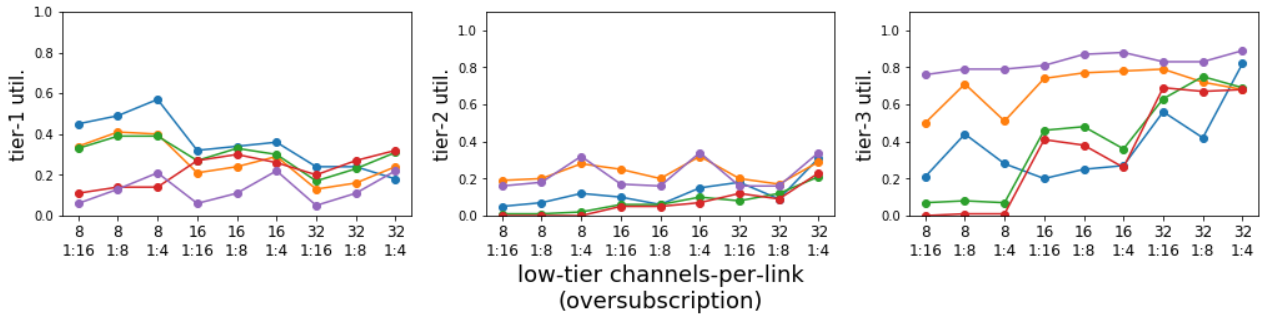

Fig. 3. blue=RL agent, orange=Tetris, green=NALB, red=NULB, purple=random. Line plots showing the utilisation of tier-1 (left), tier-2 (middle) and tier-3 (right) networking resources for each method when tested on each topology. Topology labels (x-axis) are defined as $\frac{channels\ per\ tier{-}1\ links}{oversubscription}$

shown for each topology in Fig. 4 (x-axis and y-axis respectively). In the figures, the size of a blue circle relates to the number of requests which were allocated that number of servers and distributed at that amount. Each distribution in Fig. 5 shows the combined results of allocation outcomes for all topologies of a particular oversubscription ratio (per-method). The x-axis refers to how distributed the servers allocated to a request were, and the y-axis indicates how commonly that distribution was used by an allocation method. Fig. 6 similarly shows the relationship between the resource requirement of each request (CPU and memory in x- and y- axis respectively) and how distributed those requests were (shown by colour gradient).

F.1. DRL agent uses network when it is available

Looking at the agent's results, it is seen that when the network resources are very limited (highly oversubscribed and few channels-per-link), the method concentrates allocations within racks, having higher utilisation for tier-1 (intra-rack) and comparatively very low utilisation for the higher tiers. As the agent moves towards topologies with lower oversubscription /more channels-per-link, the tier-2 and tier-3 resources are more highly utilised and the tier-1 resources less. This indicates that the agent learns a policy that exploits network resources when they are available, but allocates more rack-locally when they are not. Moreover, the biggest increase in utilisation occus at tier-3, which increases from ≈ 0.2 to ≈ 0.8 between the most and least network-resource constrained topologies. This indicates that in particular the agent learns to exploit the most non-local allocation possible (inter-cluster) when network resources are available to do so.

Observing Fig. 5, as the oversubscription moves from 1:16 (less network resources) to 1:4 (more network resources), the agent changes it's policy from being mostly intra-rack (and keeping the limited higher-tier network resources generally free) to exploiting more directly the available higher-tier network resources as they become more abundant. By contrast, the other heuristic's distributions remain comparatively static regardless of oversubcription, since the heuristics allocation scheme is hard-coded by design and not able to flexibly exploit resources in different circumstances.

F.2. The agent distributes requests differently based on their resource requirements

Fig. 4 shows that distribution is more common when more network resources are available and highly distributed requests tend to be small ones requring few servers. Fig 6 confirms

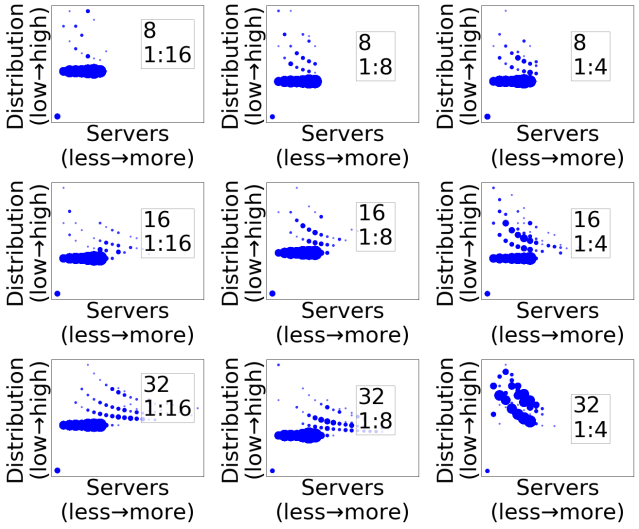

Fig. 4. Visualising the relationship between how many servers were allocated to a request (x-axis), and how distributed those servers were for that request (y-axis). Size of the blue circles represents how many requests were served at this x-y value. One figure per topology, each labelled by 2 numbers; channels-per-link at tier-1 (top) and oversubscription (bottom).

this observation of smaller requests being more likely to be highly distributed. The largest requests tend to be allocated rack-locally in all topologies and there is also a general preference for rack-locality in general. Higher-tier links in the DCN are exposed to more servers and are therefore most oversubscribed and prone to full occupation. Larger requests require more servers, and therefore more network resources to interconnect them, whereas smaller requests require fewer network resources correspondingly. Figures 4 and 6 show that the DRL agent learns to allocate minimal resources per-request from higher tiers in order to prevent congestion (full channel occupation) in these higher tier links.

This behaviour can be summarised as 1. rack-local allocations are generally preferred; 2. smaller requests have a higher likelihood of being more highly distributed; 3. larger requests have a higher likelihood of being less distributed and kept rack-local. These principles allow for the least oversubscribed network resources to be used for the requests with the most significant network resource requirements, and for smaller requests to be

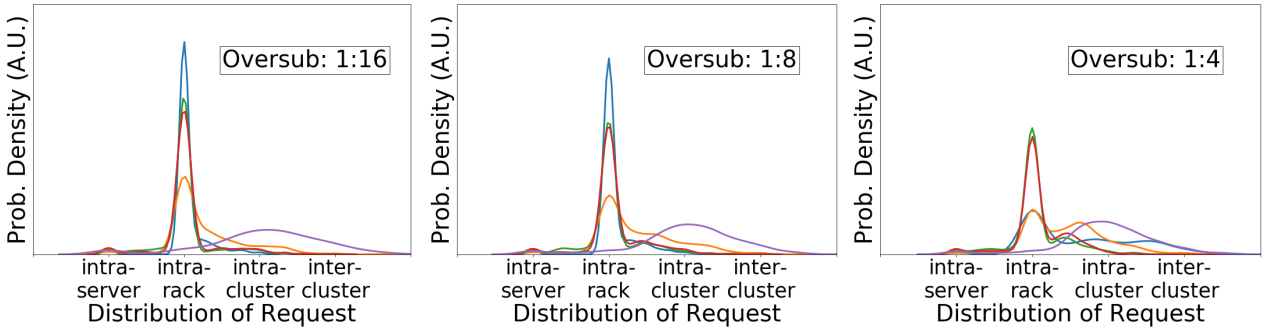

Fig. 5. blue=RL agent, orange=Tetris, green=NALB, red=NULB, purple=random. Plots showing the density of different degrees of distribution for each agent and each oversubscription ratios. Results for different topologies with the same oversubcription ratio are combined into a single graph.

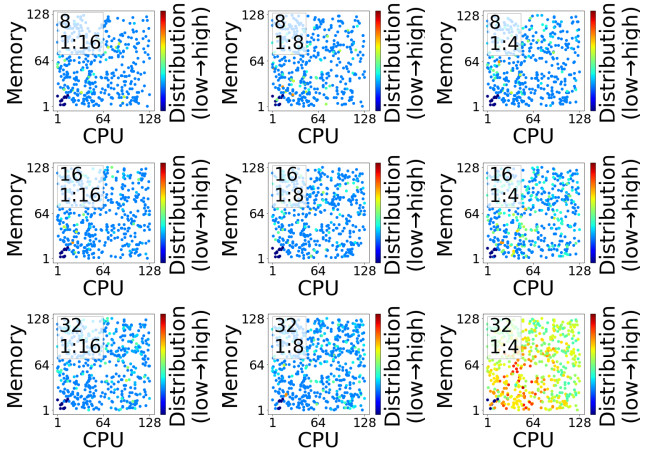

Fig. 6. Visualising the relationship between requested CPU units (x-axis), requested memory units (y-axis) and how distributed the servers allocated to that request were (colour). Lowest distribution is a single server, and maximum is inter-cluster. One figure per topology, each labelled by 2 numbers; channels-per-link at tier-1 (top) and oversubscription (bottom).

distributed more freely throughout the DCN.

6. CONCLUSION

This paper shows that deep reinforcement learning with graph neural network based policy architectures can be used to learn effective network-aware resource allocation policies end-to-end. When trained and tested across 9 data centre topologies with different network-resource quantity and oversubscription, the presented method achieves up to a 19%, 24% and 22% improvement for acceptance ratio, CPU utilisation and memory utilisation respectively against a number of baseline heuristics for network-aware resource allocation. Improvements are most pronounced when the network resources are most limited. The method also achieves the same performance as the best heuristic whilst requiring $3\times$ less network resources to do so. Additionally, the policy is highly scalable and the policy architecture topology agnostic. When trained on topologies with $\mathcal{O}(10^1)$ servers, policy performance is highly consistent when deployed on topologies with the same oversubscription properties but $\mathcal{O}(10^2)\times$ more servers with no re-training or architectural adjustments required.

Avenues of future work include; increasing the scale of test topologies beyond the $\mathcal{O}(10^3)$ shown here; handling a wider variety of/more complex request types with harder allocation constraints (latency minimums etc); more variety of possibly time-dependent request distributions to be handled during allocation rather than a single static one; stability of the policy in more dynamic topologies where servers and/or links may be randomly inaccessible (e.g. to simulate component failures).

ACKNOWLEDGEMENTS

This work was supported under the Engineering and Physical Sciences Research Council (EP/R041792/1 and EP/L015455/1), the Industrial Cooperative Awards in Science and Technology (EP/R513143/1), the OptoCloud (EP/T026081/1), TRANSNET (EP/R035342/1) grants.

DISCLOSURES

The authors do not maintain any relevant conflicts of interest.

REFERENCES

1. O. Hadary, L. Marshall, I. Menache, A. Pan, E. E. Greeff, D. Dion, S. Dorminey, S. Joshi, Y. Chen, M. Russinovich, and T. Moscibroda, "Protean: VM allocation service at scale," in *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, (USENIX Association, 2020), pp. 845–861.
2. D. A. Popescu, "Latency-driven performance in data centres," Ph.D. thesis (2019).
3. polatis.com, "Series 7000 - 384x384 port software-defined optical circuit switch," .
4. G. Zervas, F. Jiang, Q. Chen, V. Mishra, H. Yuan, K. Katrinis, D. Syrivelis, A. Reale, D. Pnevmatikatos, M. Enrico, and N. Parsons, "Disaggregated compute, memory and network systems: A new era for optical data centre architectures," in *Optical Fiber Communication Conference*, (Optical Society of America, 2017), p. W3D.4.
5. G. Zervas, H. Yuan, A. Saljoghei, Q. Chen, and V. Mishra, "Optically disaggregated data centers with minimal remote memory latency: Technologies, architectures, and resource allocation," J. Opt. Commun. Netw. **10**, A270–A285 (2018).
6. V. Mishra, J. L. Benjamin, and G. Zervas, "Monet: heterogeneous memory over optical network for large-scale data center resource disaggregation," IEEE/OSA J. Opt. Commun. Netw. **13**, 126–139 (2021).
7. F. E. Blog, *Efficient, reliable cluster management at scale with Tupperware* (2019).
8. M. Isard, V. Prabhakaran, J. Currey, U. Wieder, K. Talwar, and A. Goldberg, "Quincy: Fair scheduling for distributed computing clusters," in

Proceedings of the ACM SIGOPS 22nd Symposium on Operating Systems Principles, (Association for Computing Machinery, New York, NY, USA, 2009), SOSP '09, p. 261–276.
9. A. Verma, L. Pedrosa, M. R. Korupolu, D. Oppenheimer, E. Tune, and J. Wilkes, "Large-scale cluster management at google with borg," in *Proceedings of the European Conference on Computer Systems (EuroSys),* (Bordeaux, France, 2015).
10. M. Schwarzkopf, A. Konwinski, M. Abd-El-Malek, and J. Wilkes, "Omega: flexible, scalable schedulers for large compute clusters," in *SIGOPS European Conference on Computer Systems (EuroSys),* (Prague, Czech Republic, 2013), pp. 351–364.
11. H. Yuan, A. Saljoghei, A. Peters, and G. Zervas, "Disaggregated optical data center in a box network using parallel ocs topologies," in *Optical Fiber Communication Conference,* (Optical Society of America, 2018), p. W1C.2.
12. R. Grandl, G. Ananthanarayanan, S. Kandula, S. Rao, and A. Akella, "Multi-resource packing for cluster schedulers," in *Proceedings of the 2014 ACM Conference on SIGCOMM,* (Association for Computing Machinery, New York, NY, USA, 2014), SIGCOMM '14, p. 455–466.
13. R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction* (A Bradford Book, Cambridge, MA, USA, 2018).
14. D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. P. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," CoRR. **abs/1712.01815** (2017).
15. O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik, J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan, M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jaderberg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden, Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu, R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul, T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, and D. Silver, "Grandmaster level in StarCraft II using multi-agent reinforcement learning," Nature. **575**, 350–354 (2019).
16. P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *International Conference on Learning Representations,* (2018).
17. W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems,* vol. 30 I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds. (Curran Associates, Inc., 2017).
18. T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *5th International Conference on Learning Representations, ICLR 2017, Toulon, France, April 24-26, 2017, Conference Track Proceedings,* (OpenReview.net, 2017).
19. E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song, "Learning combinatorial optimization algorithms over graphs," in *Advances in Neural Information Processing Systems 30,* I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, eds. (Curran Associates, Inc., 2017), pp. 6348–6358.
20. R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," CoRR. **abs/1806.01973** (2018).
21. A. Mittal, A. Dhawan, S. Medya, S. Ranu, and A. Singh, "Learning heuristics over large graphs via deep reinforcement learning," (2019).
22. Z. Li, Q. Chen, and V. Koltun, "Combinatorial optimization with graph convolutional networks and guided tree search," in *Advances in Neural Information Processing Systems 31,* S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds. (Curran Associates, Inc., 2018), pp. 539–548.
23. T. D. Barrett, W. R. Clements, J. N. Foerster, and A. I. Lvovsky, "Exploratory combinatorial optimization with reinforcement learning," CoRR. **abs/1909.04063** (2019).
24. R. Addanki, S. B. Venkatakrishnan, S. Gupta, H. Mao, and M. Alizadeh, "Placeto: Learning generalizable device placement algorithms for distributed machine learning," arXiv preprint arXiv:1906.08879 (2019).
25. H. Mao, M. Schwarzkopf, S. B. Venkatakrishnan, Z. Meng, and M. Alizadeh, "Learning scheduling algorithms for data processing clusters," in *Proceedings of the ACM Special Interest Group on Data Communication,* (Association for Computing Machinery, New York, NY, USA, 2019), SIGCOMM '19, p. 270–288.
26. P. Almasan, J. Suarez-Varela, A. Badia-Sampera, K. Rusek, P. Barlet-Ros, and A. Cabello, "Deep reinforcement learning meets graph neural networks: An optical network routing use case," (2019).
27. H. Yao, X. Chen, M. Li, P. Zhang, and L. Wang, "A novel reinforcement learning algorithm for virtual network embedding," Neurocomputing **284**, 1 – 9 (2018).
28. Z. Yan, J. Ge, Y. Wu, L. Li, and T. Li, "Automatic virtual network embedding: A deep reinforcement learning approach with graph convolutional networks," IEEE J. on Sel. Areas Commun. **38**, 1040–1057 (2020).
29. A. Ghodsi, M. Zaharia, B. Hindman, A. Konwinski, S. Shenker, and I. Stoica, "Dominant resource fairness: Fair allocation of multiple resource types," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation,* (USENIX Association, USA, 2011), NSDI'11, p. 323–336.
30. A. Rai, R. Bhagwan, and S. Guha, "Generalized resource allocation for the cloud," in *Proceedings of the Third ACM Symposium on Cloud Computing,* (Association for Computing Machinery, New York, NY, USA, 2012), SoCC '12.
31. M. G. Rabbani, R. P. Esteves, M. Podlesny, G. Simon, L. Z. Granville, and R. Boutaba, "On tackling virtual data center embedding problem," in *2013 IFIP/IEEE International Symposium on Integrated Network Management (IM 2013),* (2013), pp. 177–184.
32. Z. Shabka and G. Zervas, "Nara: Learning network-aware resource allocation algorithms for cloud data centres," CoRR. **abs/2106.02412** (2021).
33. E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, J. Gonzalez, K. Goldberg, and I. Stoica, "Ray rllib: A composable and scalable reinforcement learning library," CoRR. **abs/1712.09381** (2017).
34. M. Wang, D. Zheng, Z. Ye, Q. Gan, M. Li, X. Song, J. Zhou, C. Ma, L. Yu, Y. Gai, T. Xiao, T. He, G. Karypis, J. Li, and Z. Zhang, "Deep graph library: A graph-centric, highly-performant package for graph neural networks," arXiv preprint arXiv:1909.01315 (2019).
35. T. Pohlen, B. Piot, T. Hester, M. G. Azar, D. Horgan, D. Budden, G. Barth-Maron, H. van Hasselt, J. Quan, M. Večerík, M. Hessel, R. Munos, and O. Pietquin, "Observe and look further: Achieving consistent performance on atari," (2018).
36. C. W. F. Parsonson, A. Laterre, and T. D. Barrett, "Reinforcement learning for branch-and-bound optimisation using retrospective trajectories," (2022).
37. A. Roy, H. Zeng, J. Bagga, G. Porter, and A. C. Snoeren, "Inside the social network's (datacenter) network," SIGCOMM Comput. Commun. Rev. **45**, 123–137 (2015).
38. C. W. Parsonson, J. L. Benjamin, and G. Zervas, "Traffic generation for benchmarking data centre networks," Opt. Switch. Netw. **46**, 100695 (2022).
39. B. Sharma, V. Chudnovsky, J. L. Hellerstein, R. Rifaat, and C. R. Das, "Modeling and synthesizing task placement constraints in google compute clusters," in *Proceedings of the 2nd ACM Symposium on Cloud Computing,* (2011), pp. 1–14.
40. E. Cortez, A. Bonde, A. Muzio, M. Russinovich, M. Fontoura, and R. Bianchini, "Resource central: Understanding and predicting workloads for improved resource management in large cloud platforms," in *Proceedings of the 26th Symposium on Operating Systems Principles,* (Association for Computing Machinery, New York, NY, USA, 2017), SOSP '17, p. 153–167.
41. M. Balcilar, P. Héroux, B. Gaüzère, P. Vasseur, S. Adam, and P. Honeine, "Breaking the limits of message passing graph neural networks," CoRR. **abs/2106.04319** (2021).
42. L. Poutievski *et al.*, "Jupiter evolving: Transforming google's datacenter network via optical circuit switches and software-defined networking," in *Proceedings of the ACM SIGCOMM 2022 Conference,* (Association for Computing Machinery, New York, NY, USA, 2022), SIGCOMM '22, p. 66–85.