Randomized Greedy Sampling for JSSP

Henrik Abgaryan, Ararat Harutyunyan, and Tristan Cazenave

LAMSADE, Université Paris Dauphine - PSL, CNRS, Paris, France

Abstract. The job shop scheduling problem (JSSP) is a fundamental challenge in the field of operations research and manufacturing, representing the task of optimally assigning a set of jobs to a limited number of machines to optimize one or more objectives, such as minimizing the total processing time or reducing the delay of jobs. In recent years, AI-driven methods have introduced new approaches to solving the JSSP. Continuous exploration in deep reinforcement learning (DRL) is currently concentrated on refining strategies to address the JSSP. Established DRL techniques mostly focus on better modeling and training of the Policy networks for solving JSSP problems. This paper explores the utilization of Policy networks in search algorithms. We propose two novel algorithms, Random Second Greedy Choice (RSGC) and Greedy Sampling (GS). RSGC and GS employ a randomized approach to consider alternative paths, deviating from the primary heuristic, while adjusting the probability of selecting these paths dynamically during the search process. Through experimentation, we show the effectiveness of the proposed algorithms in comparison to the usual greedy first choice inference technique and the usual sampling method.

Keywords: Job Shop Scheduling, RSGC, GS, GNN, reinforcement learning, limited discrepancy search, sampling, search algorithms

1 Introduction

The job shop scheduling problem (JSSP) stands as a pivotal challenge in operations research and manufacturing [9]. This problem entails a set of jobs, each bound by specific processing rules (like the sequential use of assigned machines), across a variety of machines. The aim here is to optimize certain parameters such as the overall completion time, workflow duration, or delay minimization.

It is proved that for JSSP instances having more than 2 machines is NP-hard [10]. Therefore, deriving precise solutions for JSSP is generally unfeasible, thereby making heuristic and approximate strategies more common for practical efficiency [6]. Traditional approaches to this problem have predominantly relied on search and inference techniques developed by the constraint programming community [1]. These techniques effectively utilize constraints to define the relationships and limitations between jobs and resources, thus enabling efficient exploration of feasible solution spaces and the identification of optimal or near-optimal schedules [13]. Another common technique is the Priority dispatching rule (PDR). It is a heuristic method that is widely used in real-world scheduling systems [17]. However, designing an effective PDR is very time-consuming, it requires solid domain knowledge for complex JSSP problems. Another approach for finding suboptimal solutions is through the help of Deep Learning and Neural Networks [3], [18]. The method based on learning can generally be categorized into two paradigms: supervised learning and reinforcement learning (RL). Ongoing research in deep reinforcement learning (DRL) is actively focused on developing new and improved methods to tackle the JSSP. Existing DRL methods represent Job Shop Scheduling Problem (JSSP) as a Markov decision process (MDP) and then learn the Policy network based on DRL. The prevailing trend leans towards refining policy networks or MDP formulation to generate better solutions. However, the exploration of effective search methods on top of the policy networks decisions and the integration of alternative strategies such as tree search, have received less attention. This indicates a potential area for further research.

In this paper, we present two search algorithms RSGC, GS, provided by pretrained Policy network inspired by [11] that utilize the categorical distribution of the pretrained Policy network to find a solution to JSSP problem. In order to check the effectiveness of the proposed search methods, we have experimented on 2 public test datasets: TA dataset [15], and DMU dataset [7]. Then we compare the results of RSGC and GS to the usual sampling algorithm and the current state-of-the-art results [18].

2 Related Work

The success of Deep Reinforcement Learning (DRL) represents a significant advancement in the field of artificial intelligence. Generally, DRL combines the principles of deep learning and reinforcement learning, enabling computers to learn complex behaviors by interacting with an environment and optimizing actions based on feedback.

In [12] the authors utilized deep Q-network (DQN) to solve a JSSP in semiconductor manufacturing plant. In [18], a new method using Deep Reinforcement Learning (DRL) is introduced to develop effective Priority Dispatching Rules (PDRs) for the Job Shop Scheduling Problem (JSSP). The approach involves formulating an MDP (Markov Decision Process) for PDR-oriented scheduling. This method utilizes a disjunctive graph representation of JSSP to capture the states, efficiently integrating operation dependencies and machine statuses for informed scheduling decisions. Additionally, the paper uses a Graph Isomorphism Network (GIN) strategy that efficiently encodes disjunctive graph nodes into fixed-dimensional embeddings. On top of the these embeddings the authors utilized Proximal Policy Optimization (PPO) algorithm [14] to train Policy network. [16] introduces a new approach for solving JSSP with DRL. It addresses the deficiencies in state representation, adding more features. This approach also models JSSP as a Markov decision process, employing a new state representation based on bidirectional scheduling. This representation allows the agent to capture more effective state information and avoid the issue of multiple optimal action selections. They also utilize the technique of Invalid Action Masking (IAM), which narrows the search space, steering the agent away from sub-optimal solutions.

Current methods are mostly based on modeling state representation and the architecture of the models. There are very few papers that use sampling or search algorithms on top of policy network to solve JSSP. A widely recognized strategy is to facilitate the step-by-step development of solutions to JSSP problems, guided by a single-shot (greedy) policy derived from a neural network, as documented in sources like [18] and [16]. These methods primarily concentrate on developing robust policy network models, aiming to elevate the quality of solutions generated in a single iteration as close to the optimal as possible. However, there is a noticeable scarcity of studies dedicated to finding effective inference methods for neural JSSP. In addition to designing and training high-quality policy networks, devising an effective inference strategy is equally crucial to maximize the quality of solutions within a specified time budget. In contrast, neural construction methods focus on generating solutions by sampling from the neural network's output probability distributions, a technique highlighted in studies such as [2]. An alternative approach to sampling is the use of Monte-Carlo Tree Search (MCTS) [4], and its variants [5]. These methods create partial solutions within a search tree using rollouts. MCTS, requires a lot of resources and long running time. Beam search represents another method used in combinatorial optimization problems [8]. Nevertheless, beam search operates on a greedy algorithm, which unconditionally adheres to the neural network's predictions, regardless of their accuracy. Like MCTS, Beam search also requires a lot of resources and long running time. Another approach called Limited Discrepancy Search (LDS) incorporates the idea of deviating from the main heuristic and sometimes selecting less promising choices [11]. It is based on the idea that sometimes, choosing solutions that don't initially seem promising might actually lead to better results. This approach acknowledges that the most obvious choice is not always the best one.

In developing our approach, we focus on a key strategy: introducing variations to the primary heuristic, specifically diverging from the initial greedy choice recommended by the policy network. This strategy lays the groundwork for our newly designed probabilistic sampling algorithms. The essence of these algorithms lies in the smart utilization of the Policy network's probability distribution also incorporating a measure of randomness to enrich decision-making processes. By integrating randomness at the beginning of our algorithm, it diverges from conventional deterministic methods, offering a nuanced way to explore the search space. It harnesses the robustness of greedy strategies while mitigating their inherent limitations through calculated randomness. This hybrid approach aims to strike a balance between the exploration and exploitation, optimizing the search process.

3 Preliminary

The Job-Shop Scheduling Problem (JSSP) is formally defined as a problem involving a set of jobs J and a set of machines M. The size of the JSSP problem instance is described as $N\_J \times N\_M$, where N_J represents the number of jobs and N_M the number of machines. For each job $J_i \in J$, it must be processed through n_i machines in a specified order $O_{i1} \rightarrow \ldots \rightarrow O_{in_i}$, where each O_{ij} (for $1 \leq j \leq n_i$) represents an operation of J_i with a processing time $p_{ij} \in \mathbb{N}$. This sequence also includes a precedence constraint. Each machine can process only one job at a time, and switching jobs mid-operation is not allowed. The objective of solving a JSSP is to determine a schedule, that is, a start time S_{ij} for each operation O_{ij}, to minimize the makespan

$C_{\max} = \max_{i,j}\{C_{ij} = S_{ij} + p_{ij}\}$ while meeting all constraints. The complexity of a JSSP instance is given by $N\_J \times N\_M$.

Furthermore, a JSSP instance can be represented through a disjunctive graph, a concept well-established in the literature [18]. Let $O = \{O_{ij}|\forall i, j\} \cup \{S, T\}$ represent the set of all operations, including two dummy operations S and T that denote the starting and ending points with zero processing time. A disjunctive graph $G = (O, C, D)$ is thus a mixed graph (a graph consisting of both directed edges (arcs) and undirected edges) with O as its vertex set. Specifically, C comprises of directed arcs (conjunctions) that represent the precedence constraints between operations within the same job, and D includes undirected arcs (disjunctions) connecting pairs of operations that require the same machine. Solving a JSSP is equivalent to determining the direction of each disjunctive arc such that the resulting graph becomes a Directed Acyclic Graph (DAG) [18]. Markov Decision Process Formulation (MDP), state representation, action space, state transition follow the methods described in [18]. An action $a_t \in A_t$ is an eligible operation at decision step t. The Policy is a neural network $\pi(a_t|s_t)$ that outputs a distribution over the actions in A_t. We use the same 2D raw features as in [18], namely a binary indicator $I(O, s_t)$ which equals to 1 only if O is scheduled in s_t, and an integer $CLB(O, s_t)$ which is the lower bound of the estimated time of completion (ETC) of O in s_t. The training has been done using Proximal Policy algortihm, which is an actor-critic algorithm [14]. "Actor" denotes the Policy network, whereas the "critic" is a distinct network that evaluates the outcomes of the decisions made by the Policy network. Both the Actor (Policy) network and the Critic network have multilayer perceptron architecture. We use the same Graph Isomorphism Network (GIN) as feature extractor, the Actor and the Critic networks as presented in [18]. GIN extracts feature embeddings of each node in an iterative and non-linear fashion. Then the Policy network uses these fixed-size embeddings for outputting a distribution over the actions. The training has been done on 20x20 training instances for 10000 steps. The training is as described in [18]. Because the Policy network is based on the fixed-sized embeddings outputted by the GIN network, it is size-agnostic which enables generalization to instances of different sizes without requiring any additional training. During the training we sample the actions according to the probability distribution outputted by the Policy network. During the inference time we utilize our search algorithms over the probability distribution outputted by the Policy network.

4 Search Algorithm

Finding the right action a_t at each state s_t is a search problem, where the Policy network $\pi(a_t|s_t)$ is used as a heuristic. When the $\pi(a_t|s_t)$ is trained and is ready for the inference, the usual method is to pick the first greedy choice action a_t, where a_t is the action with the highest probability at the state s_t, according to the categorical probability distribution of $\pi(a_t|s_t)$.

Our approach is slightly different; we formulate the problem of selecting the "correct" action a_t at each state s_t as a search problem. Let us denote by T the binary search tree. At each node n_i of the search tree T there are two options: selecting the action with the highest probability (the first greedy choice), or another action according to the probability distribution recommended by $\pi(a_t|s_t)$. Then the height of the search tree T becomes $h = N\_J \times N\_M$. At each node n_i the approach of [18] is to always choose the first greedy choice, action a_t with the highest probability at the state s_t. However as described in [11], always following the heuristic, or in our case only the first greedy choice recommended by $\pi(a_t|s_t)$, might not lead to the best possible solution. We argue that doing "discrepancies" or deviating from the first greedy choice of $\pi(a_t|s_t)$ can sometimes be better. In the context of the search trees, a "discrepancy" refers to a deviation from the most preferred or recommended path, typically represented as taking a right turn in a binary search tree that is organized based on heuristic evaluations (where the left path is usually considered the default or the more promising direction based on some criteria). There is a systematic approach of considering all of the paths in a binary search tree, as described in [11]. This systematic approach allows the search to first consider paths that are closely aligned with heuristic recommendations, and only later to explore less recommended paths. But this method can quickly become unfeasible when the search tree is large, in our case when the set of jobs J and a set of machines M is large. Our Algorithm (RSGC) Random Second Greedy Choice with Decreasing Probability gets inspiration from [11], but it also utilizes randomness. The pseudo code of our algorithm is presented in Algorithm 1. Our algorithm has two hyperparameters D_{min} and D_{max}, which are the least possible and the most possible second greedy choices we can make during the search. At each node n_i of the search tree T we compute the probability (based on the hyperparameters) of selecting the second greedy choice action. At the beginning, the probability of selecting the second greedy choice action is high. The probability then decreases as we make more and more greedy second choices. The $greedy\_2nd\_action\_count$ is the number of second greedy choices done so far. Initially, $total\_num\_greedy\_2nd\_action\_count$ is set to D_{min}. The full search is done using this fixed hyperparameter D_{min} obtaining the first makespan. Then we adjust the value of $total\_num\_greedy\_2nd\_action\_count$ by increasing it by an amount of 5% of the height h, obtaining the next makespan. We repeat this process until $total\_num\_greedy\_2nd\_action\_count$ reaches D_{max} . We then choose the best makespan.

$$probability\_of\_2nd\_action = 1 - \frac{greedy\_2nd\_action\_count}{total\_num\_greedy\_2nd\_action\_count}$$

The idea is that it is harder for the policy network to select correct actions at the beginning of the search, than at the end. Through extensive experimentation, we found that it is best to start to do around $0.25h$ second greedy choice actions. We increment the value of $0.25h$ by $0.01h$ for each subsequent iteration until we reach $0.75h$ (i.e., $D_{min} = 0.25h$ and $D_{max} = 0.75h$). This methodical approach facilitates a more effective exploration of options, thereby aiding in the identification of the optimal path characterized by the minimal completion time (*makespan*).

We also have a second similar algorithm called Greedy Sampling (GS). In GS instead of randomly selecting second probable action in line 14 of the Algorithm 1 (RSGC) we randomly sample an action according to the probability distribution recommended by the policy network $\pi(a_t|s_t)$. Note that this is not the usual sampling algorithm, as we decide whether we sample or just take the first greedy choice with probability $probability\_of\_2nd\_action$. In the Experimental section we compare the results of these two algorithms with the usual sampling algorithm.

Algorithm 1 (RSGC) Random Second Greedy Choice with Decreasing Probability

Require: $dataset$
Ensure: Results list
1: $h \leftarrow N\_J \times N\_M$
2: $D_{min} \leftarrow integer(0.25 \times h)$
3: $D_{max} \leftarrow integer(0.75 \times h)$
4: Initialize results list
5: **for** each $data$ in $dataset$ **do**
6: Initialize best makespan tracking list
7: **for** $total\_num\_greedy\_2nd\_action\_count$ in D_{min} to D_{max} **do**
8: Reset environment with $data$
9: Initialize episode reward and $greedy\_2nd\_action\_count$
10: **while** not Terminal **do**
11: $categorical\_policy\_distribution \leftarrow$ inference with policy network $\pi(a_t|s_t)$
12: $probability\_of\_2nd\_action = 1 - \frac{greedy\_2nd\_action\_count}{total\_num\_greedy\_2nd\_action\_count}$
$r \sim$ Uniform$(0, 1)$
13: **if** r $<$ $probability\_of\_2nd\_action$ **then**
14: Select the second probable action from $categorical\_policy\_distribution$
15: Increment $greedy\_2nd\_action\_count$
16: **else**
17: Greedily select the most probable action from $categorical\_policy\_distribution$
18: **end if**
19: Execute the action
20: **end while**
21: Update best makespan if current makespan is better
22: Record makespan
23: **end for**
24: Save results for current data instance
25: **end for**
26: **return** Results list

5 Experimental Results

In order to show the effectiveness of our searching algorithm, we have conducted experiments on well known TA dataset [15], and DMU dataset [7]. The Policy network is trained on instances of size $N\_J = 20$ and $N\_M = 20$. We also compare the results with those of [18]. Each run of our search algorithm makes $(D_{max} - D_{min}) \times h$ calls to the Policy network, where $h = N\_J \times N\_M$ is the height of the search tree. During the experiments we have repeated the search algorithms 10 times and noted the overall minimum and the average makespans on the corresponding tables. We compare the makespans of (RSGC) Algorithm 1 with Greedy Sampling (Ours-GS). The two algorithms just differ on line 14 of the Algorithm 1, where instead of the Greedy second choice we randomly sample an action from the probability distribution provided by the pre-trained Policy network. For the Taillard dataset instances 50×15, 50×20 and 100×20 and also for the DMU dataset 40×15, 40×20, 50×15, 50×20 we compared the best result of [18], which was inferred with Policy network trained on 30×20 instances. We have also tried to run beam search algorithm, however we had to keep many copies of the environment, and due to memory constraints it was unfeasible.

5.1 Result on Taillard's Benchmark Dataset

Table 1: Results on Taillard's Benchmark (Part I). Ours - RSGC is the result of the Algorithm 1. In Ours-GS instead of randomly selecting second probable action in line 14 of the Algorithm 1 (RSGC) we randomly sample an action according to the probability distribution recommended by the policy network $\pi(a_t|s_t)$. We repeat each experiment 10 times due to the algorithm's randomness and record the minimum and the average makespan across the 10 experiment. The "Samp (min)" and the "Samp (avg)" columns are the results of the usual sampling method. The "UB" column represents the best-known solutions from literature, with "*" indicating optimal solutions.

Instance	L2D	Ours-RS (min)	Ours-RS (avg)	Ours-RSGC (min)	Ours-RSGC (avg)	Samp (min)	Samp (avg)	UB
15x15								
Ta01	1443 (17.22%)	1401 (13.81%)	1413.4 (14.79%)	**1387 (12.66%)**	1403.9 (14.03%)	1444(17.30%)	1448.3(17.65%)	1231*
Ta02	1544 (24.12%)	1404 (12.86%)	1404.0 (12.86%)	**1361 (9.40%)**	1394 (12.06%)	1447(16.32%)	1462.0(17.52%)	1244*
Ta03	1440 (18.23%)	1420 (16.59%)	1420.6 (16.64%)	**1408 (15.60%)**	1430.1 (17.40%)	1447(18.80%)	1452.6(19.26%)	1218*
Ta04	1637 (39.32%)	**1412 (20.17%)**	1412.7 (20.23%)	1423 (21.11%)	1433.8 (22.04%)	1460(24.26%)	1484.0(26.30%)	1175*
Ta05	1619 (32.27%)	**1394 (13.89%)**	1410.0 (15.20%)	1431 (16.91%)	1435.7 (17.29%)	1437(17.40%)	1448.0(18.30%)	1224*
Ta06	1601 (29.32%)	**1392 (12.44%)**	1401.8 (13.23%)	1413 (14.12%)	1418.3 (14.55%)	1444(16.64%)	1444.6(16.69%)	1238*
Ta07	1568 (27.79%)	1402 (14.26%)	1411.5 (15.03%)	**1384 (12.78%)**	1392.3 (13.45%)	1451(18.26%)	1469.0(19.72%)	1227*
Ta08	1468 (20.62%)	**1372 (12.73%)**	1380.1 (13.39%)	1404 (15.36%)	1406.4 (15.53%)	1425(17.09%)	1427.3(17.28%)	1217*
Ta09	1627 (27.70%)	1483 (16.41%)	1491.4 (17.05%)	**1467 (15.15%)**	1469.4 (15.34%)	1544(21.19%)	1546.0(21.35%)	1274*
Ta10	1527 (23.04%)	**1401 (12.89%)**	1419.1 (14.34%)	1437 (15.79%)	1444.4 (16.39%)	1444(16.36%)	1466.6(18.18%)	1241*
20x15								
Ta11	1794 (32.19%)	**1583 (16.65%)**	1622.4 (19.57%)	1617 (19.16%)	1631.1 (20.18%)	1668(22.92%)	1684.1(24.10%)	1357*
Ta12	1805 (32.01%)	1590 (16.31%)	1595.1 (16.68%)	**1568 (14.70%)**	1573.6 (15.11%)	1658(21.29%)	1665.0(21.80%)	1367*
Ta13	1932 (43.85%)	1628 (21.22%)	1634.5 (21.69%)	**1619 (20.55%)**	1619.0 (20.55%)	1693(26.06%)	1697.0(26.36%)	1343*
Ta14	1664 (23.72%)	**1596 (18.66%)**	1600.3 (18.97%)	1604 (19.26%)	1624.7 (20.78%)	1640(21.93%)	1651.1(22.76%)	1345*
Ta15	1730 (29.20%)	**1625 (21.36%)**	1629.1 (21.65%)	1633 (21.96%)	1640.1 (22.47%)	1690(26.21%)	1694.9(26.58%)	1339*
Ta16	1710 (25.74%)	1613 (18.60%)	1641.6 (20.71%)	**1598 (17.50%)**	1610.1 (18.39%)	1693(24.49%)	1696.3(24.73%)	1360*
Ta17	1897 (29.75%)	**1728 (18.19%)**	1728.0 (18.19%)	1735 (18.66%)	1736.5 (18.76%)	1797(22.91%)	1798.0 (22.98%)	1462*
Ta18	1794 (28.51%)	**1664 (19.20%)**	1667.9 (19.49%)	1675 (20.00%)	1679.0 (20.34%)	1712(22.64%)	1712.3(22.66%)	1396
Ta19	1682 (26.28%)	**1591 (19.44%)**	1602.2 (20.29%)	1592 (19.52%)	1603.5 (20.39%)	1650(23.87%)	1653.8(24.16%)	1332*
Ta20	1739 (28.97%)	1635 (21.29%)	1635.0 (21.29%)	**1628 (20.77%)**	1657.7 (22.98%)	1666 (23.59%)	1666.3 (23.61%)	1348*
20x20								
Ta21	2252 (37.18%)	1964 (19.61%)	1973.5 (20.18%)	**1905 (16.02%)**	1914.6 (16.60%)	2021(23.08%),	2044.0(24.48%)	1642*
Ta22	2102 (31.38%)	1887 (17.94%)	1895.7 (18.48%)	**1853 (15.81%)**	1854.6 (15.91%)	1950(21.88%)	1953.7(22.11%)	1600
Ta23	2085 (33.91%)	1838 (18.05%)	1841.0 (18.24%)	**1818 (16.76%)**	1822.9 (17.05%)	1902(22.16%)	1918.8(23.24%)	1557
Ta24	2200 (33.82%)	1930 (17.40%)	1931.2 (17.48%)	**1907 (16.00%)**	1916.2 (16.54%)	1986(20.80%)	2035.3(23.80%)	1644*
Ta25	2201 (38.00%)	1918 (20.25%)	1924.4 (20.65%)	**1903 (19.31%)**	1904.6 (19.42%)	1997(25.20%)	2002.2(25.53%)	1595
Ta26	2176 (32.44%)	1961 (19.36%)	1970.4 (19.91%)	**1918 (16.73%)**	1928.9 (17.38%)	2014(22.58%)	2042.2(24.30%)	1643
Ta27	2132 (26.90%)	**2019 (20.18%)**	2028.8 (20.76%)	2026 (20.60%)	2026.0 (20.60%)	2092(24.52%)	2099.0(24.94%)	1680
Ta28	2146 (33.94%)	1883 (17.47%)	1886.6 (17.68%)	**1833 (14.35%)**	1841.5 (14.87%)	1947(21.46%)	1954.8(21.95%)	1603*
Ta29	1952 (20.12%)	1885 (16.00%)	1905.8 (17.28%)	**1877 (15.51%)**	1897.4 (16.76%)	1937(19.20%)	1984.4(22.12%)	1625
Ta30	2035 (28.47%)	**1868 (17.93%)**	1868.8 (17.98%)	1876 (18.43%)	1892.2 (19.44%)	1927(21.65%)	1934.6(22.13%)	1584
30x15								
Ta31	2565 (45.37%)	**2115 (19.91%)**	2121.0 (20.24%)	2169 (22.94%)	2183.5 (23.81%)	2170(23.02%)	2183.1(23.76%)	1764*
Ta32	2388 (33.87%)	**2212 (23.99%)**	2212.0 (23.99%)	2230 (24.94%)	2242.1 (25.65%)	2303(29.09%)	2318.0(29.93%)	1784
Ta33	2324 (29.80%)	**2204 (23.09%)**	2219.0 (23.92%)	2292 (28.03%)	2299.5 (28.45%)	2301(28.48%)	2307.8(28.86%)	1791
Ta34	2332 (27.60%)	**2207 (20.75%)**	2226.3 (21.81%)	2225 (21.74%)	2225.7 (21.79%)	2250(23.09%)	2262.7(23.78%)	1828*
Ta35	2505 (24.82%)	**2250 (12.08%)**	2260.5 (12.63%)	2256 (12.39%)	2256.0 (12.39%)	2293(14.25%)	2303.0(14.75%)	2007*
Ta36	2497 (37.31%)	**2243 (23.31%)**	2252.1 (23.83%)	2253 (23.87%)	2259.1 (24.22%)	2318(27.43%)	2320.6(27.58%)	1819*
Ta37	2325 (31.28%)	**2146 (21.20%)**	2157.1 (21.79%)	2159 (21.93%)	2168.0 (22.44%)	2181(23.15%)	2213.4(24.98%)	1771*
Ta38	2302 (37.61%)	2046 (22.30%)	2048.6 (22.45%)	**2014 (20.38%)**	2017.8 (20.63%)	2079(24.27%)	2114.9(26.41%)	1673*
Ta39	2410 (34.26%)	**2152 (19.94%)**	2152.8 (19.99%)	2160 (20.33%)	2166.8 (20.77%)	2181(21.50%)	2199.7(22.55%)	1795*
Ta40	2140 (28.21%)	2002 (19.95%)	2018.5 (20.93%)	**1989 (19.17%)**	1989.0 (19.17%)	2067(23.85%)	2083.4(24.83%)	1669

Continued on next page

Table 1 continued from previous page

Instance	L2D	Ours-GS (min)	Ours-GS (avg)	Ours-RSGC (min)	Ours-RSGC (avg)	Samp (min)	Samp (avg)	UB
30x20								
Ta41	2667 (33.02%)	**2489 (24.14%)**	2507.5 (25.06%)	2490 (24.19%)	2496.4 (24.51%)	2540(26.68%)	2568.7(28.11%)	2005
Ta42	2664 (37.53%)	2383 (23.03%)	2383.0 (23.03%)	**2358 (21.73%)**	2373.1 (22.51%)	2479(27.98%)	2490.3(28.56%)	1937
Ta43	2431 (31.69%)	2347 (27.14%)	2364.8 (28.10%)	**2332 (26.33%)**	2332.0 (26.33%)	2437(32.02%)	2457.6(33.13%)	1846
Ta44	2714 (37.14%)	2512 (26.93%)	2515.2 (27.09%)	**2471 (24.86%)**	2482.2 (25.43%)	2590(30.87%)	2595.8(31.17%)	1979
Ta45	2637 (31.85%)	2435 (21.75%)	2440.7 (22.03%)	**2380 (19.00%)**	2410.8 (20.54%)	2518(25.90%)	2530.5(26.53%)	2000
Ta46	2776 (38.38%)	2523 (25.77%)	2531.7 (26.21%)	**2435 (21.39%)**	2449.6 (22.11%)	2590(29.11%)	2595.5(29.39%)	2006
Ta47	2476 (31.07%)	2364 (25.15%)	2381.0 (26.05%)	**2344 (24.09%)**	2354.6 (24.65%)	2418(28.00%)	2429.0(28.59%)	1889
Ta48	2490 (28.55%)	2397 (23.75%)	2401.4 (24.00%)	**2339 (20.75%)**	2354.3 (21.54%)	2458(26.90%)	2462.9(27.15%)	1937
Ta49	2556 (30.34%)	2362 (20.45%)	2363.8 (20.54%)	**2349 (19.79%)**	2365.3 (20.62%)	2448(24.83%)	2459.0(25.40%)	1961
Ta50	2628 (36.66%)	**2395 (24.54%)**	2395.0 (24.54%)	2411 (25.38%)	2417.4 (25.71%)	2462(28.03%)	2463.7(28.12%)	1923
50x15								
Ta01	3599 (30.40%)	**3361 (21.78%)**	3368.2 (22.04%)	3417 (23.80%)	3417.0 (23.80%)	3404(23.33%)	3409.2(23.52%)	2760*
Ta02	3341 (21.23%)	**3212 (16.55%)**	3212.0 (16.55%)	3249 (17.89%)	3270.0 (18.65%)	3304(19.88%)	3313.8(20.24%)	2756*
Ta03	3186 (17.26%)	**3000 (10.42%)**	3004.1 (10.57%)	3045 (12.07%)	3050.1 (12.26%)	3084(13.51%)	3091.6(13.79%)	2717*
Ta04	3266 (15.04%)	**3120 (9.90%)**	3132.6 (10.34%)	3134 (10.39%)	3134.0 (10.39%)	3183(12.12%)	3201.0(12.75%)	2839*
Ta05	3232 (20.64%)	**3132 (16.91%)**	3140.1 (17.21%)	3111 (16.13%)	3124.8 (16.64%)	3196(19.30%)	3226.3(20.43%)	2679*
Ta06	3378 (21.47%)	**3134 (12.69%)**	3146.7 (13.15%)	3169 (13.95%)	3175.4 (14.18%)	3197(14.96%)	3206.4(15.30%)	2781*
Ta07	3471 (17.94%)	**3310 (12.47%)**	3316.6 (12.69%)	3349 (13.80%)	3353.6 (13.95%)	3340(13.49%)	3357.1(14.07%)	2943*
Ta08	3732 (29.36%)	3289 (14.00%)	3296.6 (14.27%)	**3276 (13.55%)**	3300.3 (14.40%)	3350(16.12%)	3352.9(16.22%)	2885*
Ta09	3381 (27.34%)	3131 (17.93%)	3136.8 (18.15%)	**3129 (17.85%)**	3129.0 (17.85%)	3151(18.68%)	3174.4(19.56%)	2655*
Ta10	3352 (23.10%)	**3035 (11.46%)**	3045.5 (11.84%)	3065 (12.56%)	3067.7 (12.66%)	3076(12.96%)	3078.7(13.06%)	2723*
50x20								
Ta11	3654 (27.40%)	**3447 (20.18%)**	3464.6 (20.78%)	3472 (21.04%)	3474.5 (21.11%)	3534(23.22%)	3551.2(23.82%)	2868*
Ta12	3722 (29.73%)	3419 (19.16%)	3461.7 (20.65%)	**3408 (18.78%)**	3411.7 (18.92%)	3544(23.53%)	3555.3(23.92%)	2869*
Ta13	3536 (28.32%)	**3143 (14.08%)**	3172.7 (15.17%)	3147 (14.23%)	3165.5 (14.88%)	3253(18.08%)	3257.2(18.23%)	2755*
Ta14	3631 (34.39%)	3143 (16.32%)	3149.1 (16.54%)	**3132 (15.91%)**	3135.2 (16.03%)	3239(19.87%)	3239.8(19.90%)	2702*
Ta15	3359 (23.28%)	3233 (18.67%)	3242.0 (18.98%)	**3186 (16.92%)**	3193.3 (17.19%)	3325(22.02%)	3338.4(22.51%)	2725*
Ta16	3555 (24.96%)	3296 (15.85%)	3304.1 (16.14%)	**3270 (14.94%)**	3275.5 (15.13%)	3374(18.59%)	3393.2(19.27%)	2845*
Ta17	3567 (26.27%)	3423 (21.17%)	3428.3 (21.36%)	**3384 (19.79%)**	3403.6 (20.48%)	3463(22.58%)	3490.4(23.55%)	2825*
Ta18	3680 (32.18%)	**3168 (13.79%)**	3168.0 (13.79%)	3178 (14.15%)	3196.9 (14.83%)	3300(18.53%)	3304.7(18.70%)	2784*
Ta19	3592 (16.97%)	3482 (13.38%)	3494.7 (13.80%)	**3480 (13.32%)**	3505.8 (14.16%)	3559(15.89%)	3560.5(15.94%)	3071*
Ta20	3643 (21.64%)	3551(18.56%)	3601 (20.23%)	**3482 (16.26%)**	3490.8 (16.55%)	3596(20.07%)	3599.7(20.19%)	2995*
100x20								
Ta21	6452 (18.08%)	6049 (10.71%)	6057.0 (10.85%)	**6023 (10.23%)**	6036.0 (10.47%)	6064(10.98%)	6084.2(11.35%)	5464*
Ta22	5695 (9.92%)	5609 (8.26%)	5615.0 (8.38%)	**5540 (6.93%)**	5540.6 (6.94%)	5667(9.38%)	5698.1(9.98%)	5181*
Ta23	6462 (16.06%)	6148 (10.42%)	6168.4 (10.78%)	**6109 (9.72%)**	6109.0 (9.72%)	6139(10.26%)	6139.0(10.26%)	5568*
Ta24	5885 (10.23%)	**5717 (7.08%)**	5731.0 (7.34%)	5749 (7.68%)	5749.0 (7.68%)	5775(8.17%)	5780.0(8.26%)	5339*
Ta25	6355 (17.86%)	6167 (14.37%)	6174.4 (14.51%)	6247 (15.86%)	6276.8 (16.41%)	**6137(13.82%)**	6177.1(14.56%)	5392*
Ta26	6135 (14.84%)	**5890 (10.26%)**	5890.0 (10.26%)	5905 (10.54%)	5932.0 (11.04%)	5909(10.61%)	6019.2(12.68%)	5342*
Ta27	6056 (11.41%)	**5768 (6.11%)**	5791.2 (6.53%)	5777 (6.27%)	5799.2 (6.68%)	5824(7.14%)	5879.0(8.15%)	5436*
Ta28	6101 (13.11%)	5924 (9.83%)	5926.0 (9.86%)	5984 (10.94%)	5985.4 (10.96%)	**5920(9.75%)**	5958.0(10.46%)	5394*
Ta29	5943 (10.92%)	5782 (7.91%)	5782.0 (7.91%)	**5738 (7.09%)**	5747.4 (7.27%)	5839(8.98%)	5850.0(9.18%)	5358*
Ta30	5892 (13.68%)	**5692 (9.82%)**	5705.2 (10.08%)	5695 (9.88%)	5705.0 (10.07%)	5707(10.11%)	5717.0(10.30%)	5183*

Table 2: Results on DMU's Benchmark (Part I). Ours - RSGC is the result of the Algorithm 1. In Ours-GS instead of randomly selecting second probable action in line 14 of the Algorithm 1 (RSGC) we randomly sample an action according to the probability distribution recommended by the policy network $\pi(a_t|s_t)$. We repeat each experiment 10 times due to the algorithm's randomness and record the minimum and the average makespan across the 10 experiment. The "Samp (min)" and the "Samp (avg)" columns are the results of the usual sampling method. The "UB" column represents the best-known solutions from literature, with "*" indicating optimal solutions.

Instance	L2D	Ours-GS (min)	Ours-GS (avg)	Ours-RSGC (min)	Ours-RSGC (avg)	Samp (min)	Samp (avg)	UB
20x15								
Dmu01	3323 (29.65%)	**3132 (22.20%)**	3132.0 (22.20%)	3198 (24.78%)	3204.7 (25.04%)	3218(25.56%)	3270.4(27.60%)	2563
Dmu02	3630 (34.15%)	3269 (20.81%)	3302.5 (22.04%)	**3266 (20.69%)**	3270.8 (20.87%)	3360(24.17%)	3363.8(24.31%)	2706
Dmu03	3660 (34.02%)	**3287 (20.36%)**	3287.0 (20.36%)	3375 (23.58%)	3384.4 (23.93%)	3392(24.20%)	3429.9(25.59%)	2731*
Dmu04	3816 (42.97%)	**3171 (18.81%)**	3192.9 (19.63%)	3261 (22.18%)	3267.4 (22.42%)	3229(20.98%)	3286.7(23.14%)	2669
Dmu05	3897 (41.76%)	**3377 (22.84%)**	3392.0 (23.39%)	3394 (23.46%)	3394.0 (23.46%)	3465(26.05%)	3491.5(27.01%)	2749*
Dmu41	4316 (32.88%)	4070 (25.31%)	4085.2 (25.78%)	**4050 (24.69%)**	4070.0 (25.31%)	4250(30.85%)	4256.8(31.06%)	3248
Dmu42	4858 (43.30%)	**4493 (32.54%)**	4493.0 (32.54%)	4526 (33.51%)	4541.3 (33.96%)	4650(37.17%)	4654.6(37.30%)	3390
Dmu43	4887 (42.02%)	**4373 (27.09%)**	4373.0 (27.09%)	4445 (29.18%)	4455.1 (29.47%)	4531(31.68%)	4558.4(32.47%)	3441

Continued on next page

Table 2 continued from previous page

Instance	L2D	Ours-GS (min)	Ours-GS (avg)	Ours-RSGC (min)	Ours-RSGC (avg)	Samp (min)	Samp (avg)	UB
Dmu44	5151 (47.68%)	**4592 (31.65%)**	4604.9 (32.02%)	4637 (32.94%)	4637.0 (32.94%)	4692(34.52%)	4807.5(37.83%)	3488
Dmu45	4615 (41.05%)	**4377 (33.77%)**	4387.4 (34.09%)	4387 (34.08%)	4399.6 (34.46%)	4454(36.12%)	4472.4(36.69%)	3272
20x20								
Dmu06	4358 (34.34%)	3880 (19.61%)	3890.2 (19.92%)	**3844 (18.50%)**	3876.0 (19.48%)	3977(22.60%)	4011.5(23.66%)	3244
Dmu07	3671 (20.51%)	3621 (18.88%)	3640.3 (19.51%)	**3621 (18.88%)**	3637.8 (19.43%)	3733(22.55%)	3750.2(23.12%)	3046
Dmu08	4048 (26.98%)	**3726 (16.88%)**	3760.2 (17.95%)	3730 (17.00%)	3755.1 (17.79%)	3882(21.77%)	3946.3(23.79%)	3188
Dmu09	4482 (44.95%)	3790 (22.57%)	3817.8 (23.47%)	**3695 (19.50%)**	3695.0 (19.50%)	3893(25.91%)	3925.0(26.94%)	3092
Dmu10	4021 (34.75%)	**3494 (17.09%)**	3551.9 (19.03%)	3519 (17.93%)	3550.2 (18.97%)	3578(19.91%)	3632.5(21.73%)	2984
Dmu46	5876 (45.63%)	5066 (25.55%)	5066.0 (25.55%)	**5063 (25.48%)**	5085.5 (26.03%)	5310(31.60%)	5314.3(31.71%)	4035
Dmu47	5771 (46.51%)	4930 (25.16%)	4965.7 (26.06%)	**4872 (23.69%)**	4962.5 (25.98%)	5177(31.43%)	5227.6(32.71%)	3939
Dmu48	5034 (33.78%)	4725 (25.56%)	4741.0 (25.99%)	**4640 (23.31%)**	4667.0 (24.02%)	4891(29.98%)	4906.5(30.39%)	3763
Dmu49	5470 (47.44%)	4671 (25.90%)	4683.3 (26.23%)	**4731 (27.52%)**	4738.8 (27.73%)	4952(33.48%)	4984.1(34.34%)	3710
Dmu50	5314 (42.50%)	4862 (30.38%)	4880.1 (30.87%)	**4837 (29.71%)**	4845.5 (29.94%)	5052(35.48%)	5088.6(36.46%)	3729
30x15								
Dmu11	4435 (29.30%)	**4183 (21.95%)**	4194.5 (22.29%)	4232 (23.38%)	4236.5 (23.51%)	4282(24.84%)	4285.3(24.94%)	3430
Dmu12	4864 (39.17%)	4247 (21.52%)	4260.3 (21.90%)	**4247 (21.52%)**	4254.5 (21.73%)	4414(26.29%)	4431.4(26.79%)	3495
Dmu13	4918 (33.60%)	4486 (21.87%)	4515.2 (22.66%)	**4457 (21.08%)**	4478.5 (21.67%)	4613(25.32%)	4641.5(26.09%)	3681*
Dmu14	4130 (21.69%)	3969 (16.94%)	3977.7 (17.20%)	**3930 (15.79%)**	3933.5 (15.90%)	4085(20.36%)	4097.5(20.73%)	3394*
Dmu15	4392 (31.38%)	**4054 (21.27%)**	4064.0 (21.57%)	4072 (21.81%)	4076.4 (21.94%)	4114(23.06%)	4138.1(23.78%)	3343*
Dmu51	6241 (49.77%)	**5919 (42.04%)**	5951.7 (42.83%)	6090 (46.15%)	6095.2 (46.27%)	6093(46.22%)	6128.1(47.06%)	4167
Dmu52	6714 (55.74%)	**6032 (39.92%)**	6093.2 (41.34%)	6081 (41.06%)	6172.6 (43.18%)	6235(44.63%)	6266.0(45.35%)	4311
Dmu53	6724 (53.03%)	**6010 (36.78%)**	6096.0 (38.73%)	6163 (40.26%)	6187.8 (40.82%)	6287(43.08%)	6308.5(43.57%)	4394
Dmu54	6522 (49.52%)	**6034 (38.33%)**	6050.8 (38.72%)	6072 (39.20%)	6072.0 (39.20%)	6201(42.16%)	6234.9(42.94%)	4362
Dmu55	6639 (55.44%)	**5917 (38.54%)**	5917.0 (38.54%)	5931 (38.87%)	5945.2 (39.20%)	6031(41.21%)	6042.4(41.48%)	4271
30x20								
Dmu16	4593 (32.04%)	**4462 (18.95%)**	4656.7 (24.15%)	4696 (25.19%)	4713.5 (25.66%)	4773(27.24%)	4801.6(28.00%)	3751
Dmu17	5379 (41.03%)	4747 (24.46%)	4751.0 (24.57%)	**4710 (23.49%)**	4736.9 (24.20%)	4907(28.65%)	4930.0(29.26%)	3814
Dmu18	5100 (32.67%)	4639 (20.68%)	4651.8 (21.01%)	**4546 (18.26%)**	4564.4 (18.74%)	4804(24.97%)	4834.9(25.77%)	3844*
Dmu19	4889 (29.75%)	4659 (23.65%)	4668.5 (23.90%)	**4651 (23.43%)**	4678.5 (24.16%)	4780(26.85%)	4792.8(27.19%)	3768
Dmu20	4859 (30.97%)	4442 (19.73%)	4489.7 (21.02%)	**4388 (18.27%)**	4441.5 (19.72%)	4554(22.75%)	4628.2(24.75%)	3710
Dmu56	7328 (48.31%)	**6784 (37.30%)**	6784.0 (37.30%)	6861 (38.86%)	6861.7 (38.87%)	6924(40.13%)	6998.7(41.65%)	4941
Dmu57	6704 (44.02%)	**6421 (37.94%)**	6435.0 (38.24%)	6428 (38.09%)	6509.0 (39.83%)	6625(42.32%)	6638.0(42.60%)	4655
Dmu58	6721 (42.76%)	**6322 (34.28%)**	6353.7 (34.96%)	6398 (35.90%)	6427.9 (36.53%)	6564(39.42%)	6584.0(39.85%)	4708
Dmu59	7109 (53.74%)	**6388 (38.15%)**	6420.2 (38.85%)	6486 (40.27%)	6492.4 (40.41%)	6398(38.37%)	6579.3(42.29%)	4624
Dmu60	6632 (39.47%)	**6400 (34.60%)**	6471.4 (36.10%)	6459 (35.84%)	6465.4 (35.97%)	6631(39.45%)	6665.5(40.18%)	4755
40x15								
Dmu21	5317 (21.42%)	**5034 (14.91%)**	5069.7 (15.75%)	5072 (15.82%)	5077.8 (15.95%)	5162(17.85%)	5235.3(19.53%)	4380*
Dmu22	5534 (17.12%)	**5293 (12.03%)**	5360.3 (13.45%)	5316 (12.51%)	5331.9 (12.85%)	5432(14.96%)	5475.5(15.88%)	4725*
Dmu23	5620 (20.41%)	**5171 (10.78%)**	5195.8 (11.30%)	5178 (10.92%)	5179.4 (10.94%)	5291(13.35%)	5303.3(13.61%)	4668*
Dmu24	5753 (23.77%)	**5138 (10.54%)**	5138.0 (10.54%)	5224 (12.41%)	5236.5 (12.68%)	5324(14.54%)	5353.5(15.18%)	4648*
Dmu25	4775 (14.66%)	4659 (11.89%)	4662.2 (11.97%)	**4618 (10.90%)**	4618.0 (10.90%)	4755(14.19%)	4756.4(14.23%)	4164*
Dmu61	8203 (58.62%)	**7634 (47.58%)**	7685.5 (48.54%)	7818 (51.14%)	7829.2 (51.37%)	7802(50.85%)	7829.1(51.37%)	5172
Dmu62	8091 (53.66%)	**7528 (42.96%)**	7561.4 (43.59%)	7810 (48.32%)	7810.0 (48.32%)	7768(47.54%)	7799.9(48.15%)	5265
Dmu63	8031 (50.76%)	**7543 (41.66%)**	7597.2 (42.69%)	7610 (42.92%)	7621.2 (43.15%)	7636(43.37%)	7744.8(45.41%)	5326
Dmu64	7738 (47.39%)	**7628 (45.20%)**	7670.0 (46.10%)	7757 (47.66%)	7773.5 (47.97%)	7717(46.99%)	7797.6(48.53%)	5250
Dmu65	7577 (46.07%)	**7345 (41.58%)**	7370.5 (42.00%)	7648 (47.40%)	7648.0 (47.40%)	7583(46.11%)	7584.0(46.13%)	5190
40x20								
Dmu26	5946 (27.94%)	5646 (21.51%)	5662.9 (21.86%)	**5583 (20.14%)**	5592.2 (20.34%)	5769(24.14%)	5823.2(25.31%)	4647*
Dmu27	6418 (32.38%)	5874 (21.14%)	5902.2 (21.74%)	**5849 (20.63%)**	5851.7 (20.68%)	6014(24.05%)	6046.4(24.72%)	4848*
Dmu28	5986 (27.57%)	5610 (19.56%)	5622.8 (19.81%)	**5604 (19.43%)**	5609.8 (19.54%)	5831(24.28%)	5848.5(24.65%)	4692*
Dmu29	6051 (29.00%)	5776 (23.09%)	5776.0 (23.09%)	**5685 (21.14%)**	5685.0 (21.14%)	5928(26.37%)	5942.8(26.69%)	4691*
Dmu30	5988 (26.58%)	**5584 (18.01%)**	5676.0 (19.95%)	5718 (20.85%)	5718.0 (20.85%)	5791(22.38%)	5847.2(23.57%)	4732*
Dmu66	8475 (48.24%)	**8069 (41.14%)**	8106.8 (41.80%)	8099 (41.67%)	8129.0 (42.19%)	8224(43.85%)	8265.6(44.57%)	5717
Dmu67	8832 (51.94%)	**8227 (41.53%)**	8259.6 (42.09%)	8252 (41.96%)	8318.0 (43.09%)	8361(43.83%)	8377.3(44.11%)	5813
Dmu68	8693 (50.58%)	**8198 (42.01%)**	8259.6 (43.07%)	8364 (44.88%)	8432.7 (46.07%)	8348(44.60%)	8348.0(44.60%)	5773
Dmu69	8634 (51.23%)	**8107 (42.00%)**	8159.4 (42.92%)	8202 (43.67%)	8228.3 (44.13%)	8209(43.79%)	8240.7(44.34%)	5709
Dmu70	8735 (48.33%)	8341 (41.64%)	8375.2 (42.22%)	**8230 (39.75%)**	8244.0 (39.99%)	8416(42.91%)	8531.2(44.86%)	5889
50x15								
Dmu31	7156 (26.88%)	**6400 (13.48%)**	6423.2 (13.89%)	6512 (15.48%)	6524.3 (15.70%)	6552(16.17%)	6603.6(17.09%)	5640*
Dmu32	6506 (9.76%)	**6025 (1.65%)**	6032.9 (1.78%)	6168 (4.06%)	6175.2 (4.18%)	6133(3.48%)	6137.6(3.55%)	5927*
Dmu33	6192 (8.08%)	6001 (4.75%)	6016.9 (4.97%)	**5898 (2.96%)**	5929.1 (3.51%)	6015(5.01%)	6117.5(6.80%)	5728*
Dmu34	6257 (16.18%)	5948 (10.47%)	5954.1 (10.57%)	**5868 (8.95%)**	5892.4 (9.41%)	6133(13.89%)	6169.5(14.57%)	5385*
Dmu35	6302 (11.83%)	6012 (6.70%)	6021.3 (6.85%)	**6012 (6.70%)**	6015.2 (6.74%)	6084(7.97%)	6129.8(8.78%)	5635*
Dmu71	9797 (57.24%)	**9440 (51.47%)**	9461.6 (51.81%)	9521 (52.78%)	9526.9 (52.88%)	9571(53.55%)	9585.6(53.79%)	6233
Dmu72	9926 (53.11%)	**9681 (49.33%)**	9681.0 (49.33%)	9650 (48.85%)	9670.9 (49.17%)	9688(49.44%)	9710.2(49.78%)	6483

Continued on next page

Table 2 continued from previous page

Instance	L2D	Ours-GS (min)	Ours-GS (avg)	Ours-RSGC (min)	Ours-RSGC (avg)	Samp (min)	Samp (avg)	UB
Dmu73	9933 (61.17%)	**9298 (50.87%)**	9365.8 (51.97%)	9545 (54.88%)	9545.0 (54.88%)	9447(53.29%)	9469.5(53.65%)	6163
Dmu74	9833 (58.09%)	**9440 (51.77%)**	9461.0 (52.11%)	9607 (54.45%)	9618.0 (54.63%)	9560(53.70%)	9581.0(54.04%)	6220
Dmu75	9892 (59.63%)	**9287 (49.86%)**	9300.9 (50.09%)	9299 (50.06%)	9307.0 (50.19%)	9397(51.64%)	9448.0(52.46%)	6197
50x20								
Dmu36	7470 (32.89%)	**6665 (18.57%)**	6745.3 (20.00%)	6787 (20.74%)	6790.5 (20.81%)	6825(21.42%)	6840.6(21.70%)	5621*
Dmu37	7296 (24.70%)	**6843 (16.95%)**	6873.0 (17.47%)	6866 (17.35%)	6897.9 (17.89%)	6895(17.84%)	6927.4(18.40%)	5851*
Dmu38	7410 (29.70%)	**6968 (21.97%)**	7012.0 (22.74%)	7136 (24.91%)	7149.7 (25.15%)	7075(23.84%)	7079.8(23.92%)	5713*
Dmu39	6827 (18.79%)	6580 (14.49%)	6601.6 (14.87%)	**6434 (11.95%)**	6443.8 (12.12%)	6654(15.78%)	6706.7(16.70%)	5747*
Dmu40	7325 (31.34%)	**6718 (20.46%)**	6771.7 (21.42%)	6773 (21.45%)	6776.1 (21.50%)	6835(22.56%)	6841.6(22.68%)	5577*
Dmu76	**9698 (42.35%)**	9823 (44.18%)	9851.4 (44.60%)	9991 (46.65%)	9997.3 (46.74%)	9922(45.63%)	9963.5(46.24%)	6813
Dmu77	10693 (56.74%)	**9930 (45.56%)**	9940.8 (45.72%)	9948 (45.82%)	9974.4 (46.21%)	10070(47.61%)	10070.0(47.61%)	6822
Dmu78	9986 (47.50%)	**9940 (46.82%)**	10013.6 (47.91%)	9990 (47.56%)	10029.2 (48.14%)	10126(49.57%)	10151.5(49.95%)	6770
Dmu79	10936 (56.90%)	10303 (47.82%)	10310.6 (47.93%)	**10181 (46.07%)**	10181.0 (46.07%)	10457(50.03%)	10486.5(50.45%)	6970
Dmu80	9875 (47.70%)	**9674 (44.69%)**	9742.9 (45.73%)	9859 (47.46%)	9874.8 (47.69%)	9732(45.56%)	9803.5(46.3%)	6686

6 Conclusion

In this study, we present RSGC and GS, algorithms designed to optimize JSSP through Policy network-guided search. By incorporating second greedy choices and policy network sampling methods with decreasing probabilities RSGC offers a balance between exploration and exploitation in the search space, thereby mitigating the risk of finding solutions having larger makespan. Our systematic approach to adjusting the probability of selecting alternative paths enables efficient exploration of the search tree, leading to improved makespan. Through extensive experimentation, we establish optimal parameters for RSGC and GS, enhancing its performance across two benchmark datasets. We also compare the minimum makespans and the average makespans using the two algorithms. The experiments showed that the GS algorithm results were slightly better on most instances, the reason perhaps being that it explored more. Overall, RSGC presents a promising avenue for enhancing scheduling algorithms, showcasing the potential of Policy networks in addressing complex optimization challenges.

References

1. J. Christopher Beck, T. K. Feng, and Jean-Paul Watson. Combining constraint programming and local search for job-shop scheduling. *INFORMS Journal on Computing*, 23(1):1–14, 2010.
2. Irwan Bello, Hieu Pham, Quoc V. Le, Mohammad Norouzi, and Samy Bengio. Neural combinatorial optimization with reinforcement learning, 2017.
3. Giovanni Bonetta, Davide Zago, Rossella Cancelliere, and Andrea Grosso. Job shop scheduling via deep reinforcement learning: a sequence to sequence approach. *Not Specified*, Aug 2023.
4. Cameron B. Browne, Edward Powley, Daniel Whitehouse, Simon M. Lucas, Peter I. Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in Games*, 4(1):1–43, 2012.
5. Tristan Cazenave. Nested Monte-Carlo search. In *Proceedings of the IJCAI International Joint Conference on Artificial Intelligence*, pages 456–461, 2009.
6. Ceren Cebi, Enes Atac, and Ozgur Koray Sahingoz. Job shop scheduling problem and solution algorithms: A review. In *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–7, 2020.

7. Ebru Demirkol, Sanjay Mehta, and Reha Uzsoy. Benchmarks for shop scheduling problems. *European Journal of Operational Research*, 109(1):137–141, 1998.
8. Rupert Ettrich, Marc Huber, and Günther Raidl. *A Policy-Based Learning Beam Search for Combinatorial Optimization*, pages 130–145. Springer, 03 2023.
9. Kaizhou Gao, Zhiguang Cao, Le Zhang, Zhenghua Chen, Yuyan Han, and Quanke Pan. A review on swarm intelligence and evolutionary algorithms for solving flexible job shop scheduling problems. *IEEE/CAA Journal of Automatica Sinica*, 6(4):904, 2019.
10. Michael R Garey, David S Johnson, and Ravi Sethi. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research*, 1(2):117–129, 1976.
11. William D Harvey and Matthew L Ginsberg. Limited discrepancy search. In *IJCAI*, pages 607–615, 1995.
12. Chun-Cheng Lin, Der-Jiunn Deng, Yen-Ling Chih, and Hsin-Ting Chiu. Smart manufacturing scheduling with edge computing using multiclass deep q network. *IEEE Transactions on Industrial Informatics*, 15(7):4276–4284, 2019.
13. Eugeniusz Nowicki and Czeslaw Smutnicki. An advanced tabu search algorithm for the job shop problem. *Journal of Scheduling*, 8(2):145–159, 2005.
14. John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.
15. Eric Taillard. Benchmarks for basic scheduling problems. *European Journal of Operational Research*, 64(2):278–285, 1993.
16. Erdong Yuan, Shuli Cheng, Liejun Wang, Shiji Song, and Fang Wu. Solving job shop scheduling problems via deep reinforcement learning. *Applied Soft Computing*, 143:110436, 2023.
17. Mohamed Habib Zahmani, Baghdad Atmani, Abdelghani Bekrar, and Nassima Aissani. Multiple priority dispatching rules for the job shop scheduling problem. In *3rd International Conference on Control, Engineering Information Technology (CEIT'2015)*, Tlemcen, Algeria, 2015.
18. Cong Zhang, Wen Song, Zhiguang Cao, Jie Zhang, Puay Siew Tan, and Chi Xu. Learning to dispatch for job shop scheduling via deep reinforcement learning. In *34th Conference on Neural Information Processing Systems (NeurIPS)*, 2020.