Pamukkale Univ Muh Bilim Derg, 32(2), 235-246, 2026

Pamukkale Üniversitesi Mühendislik Bilimleri Dergisi

Pamukkale University Journal of Engineering Sciences

A new neighborhood-based VNS algorithm for unrelated parallel machine scheduling problem with sequence-dependent setup times

Sıra bağımlı hazırlık süreli ilişkisiz paralel makine çizelgeleme problemi için yeni bir komşuluk tabanlı DKA algoritması

Günay Kılıç[1*], *Arzu Organ*[2]

[1]Pamukkale University Rectorate, Denizli, Türkiye.
gkilic@pau.edu.tr

[2]Department of Business Administration, Faculty of Economics and Administrative Sciences, Pamukkale Uni., Denizli, Türkiye.
aorgan@pau.edu.tr

Received/Geliş Tarihi: 11.02.2025 Revision/Düzeltme Tarihi: 08.08.2025 doi: 10.5505/pajes.2025.05935
Accepted/Kabul Tarihi: 13.08.2025 Research Article/Araştırma Makalesi

Abstract

Scheduling involves the allocation of tasks to machines under specific constraints and criteria. As schedules are constructed and jobs are assigned to machines, various scheduling challenges arise. This study focuses on the NP-hard Unrelated Parallel Machine Scheduling Problem with Sequence-Dependent Setup Times (UPMSPSDT), where jobs have varying processing times across machines, and setup times between jobs depend on the machine. The objective is to minimize the makespan (Cmax) of the final schedule. Due to its computational complexity, exact methods are ineffective in solving UPMSPSDT efficiently, leading researchers to explore metaheuristic approaches for near-optimal solutions. This study aims to enhance solution quality for UPMSPSDT using the Variable Neighborhood Search (VNS) algorithm, a single-solution metaheuristic. To this end, a novel neighborhood structure is introduced, inspired by the shell-changing behavior of crabs, complementing existing structures in literature. Additionally, three different local search strategies are evaluated based on the makespan values obtained through neighborhood transitions and are applied iteratively using a local search selection strategy. Furthermore, an improved version of a greedy initial solution from the literature is proposed to generate higher-quality starting solutions. The proposed Crab-inspired Neighborhood-based VNS (CNVNS) is tested on a widely used benchmark dataset, and the results are analyzed. Findings indicate that the proposed algorithm outperforms benchmarked approaches in achieving lower Cmax values, demonstrating its effectiveness in solving UPMSPSDT.

Keywords: Variable neighborhood search, Metaheuristic, Unrelated parallel machine scheduling problem with sequence-dependent setup times.

Öz

Çizelgeleme, belirli kısıtlar ve kriterler doğrultusunda görevlerin makinelere tahsis edilmesi sürecidir. Çizelgeler oluşturulup işler makinelere atandıkça çeşitli çizelgeleme problemleri ortaya çıkmaktadır. Bu çalışma, işler farklı makinelere göre değişen işlem sürelerine sahipken ve işler arasındaki hazırlık süreleri makineye bağlı olarak değişirken, nihai tamamlanma süresinin (Cmax) en aza indirilmesini amaçlayan Sıra Bağımlı Hazırlık Süreli İlişkili Olmayan Paralel Makine Çizelgeleme Problemi (UPMSPSDT) üzerine odaklanmaktadır. Hesaplama açısından NP-zor bir problem olması nedeniyle, kesin yöntemler UPMSPSDT'yi etkin bir şekilde çözmede yetersiz kalmakta ve bu nedenle araştırmacılar yaklaşık çözümler elde etmek için metasezgisel yaklaşımlara yönelmektedir. Bu çalışmada, Değişken Komşuluk Arama (DKA) algoritması kullanılarak UPMSPSDT için daha kaliteli çözümler elde edilmesi amaçlanmaktadır. Bu doğrultuda, yengeçlerin kabuk değiştirme davranışından esinlenen ve literatürde mevcut komşuluk yapılarını tamamlayıcı nitelikte yeni bir komşuluk yapısı önerilmektedir. Ayrıca, üç farklı yerel arama yöntemi, komşuluk değişimlerinden elde edilen tamamlanma süresi (Cmax) değerlerine göre değerlendirilmiş ve yerel arama seçim stratejisi kullanılarak iteratif olarak uygulanmıştır. Bunun yanı sıra, literatürdeki açgözlü (greedy) başlangıç çözümüne yönelik bir iyileştirme önerilerek, daha yüksek kaliteli başlangıç çözümlerinin elde edilmesi hedeflenmiştir. Önerilen Yengeç İlhamlı Komşuluk Tabanlı DKA (CNVNS) algoritması, yaygın olarak kullanılan bir test veri seti üzerinde değerlendirilmiş ve sonuçlar analiz edilmiştir. Elde edilen bulgular, önerilen algoritmanın karşılaştırıldığı diğer yaklaşımlara kıyasla daha düşük Cmax değerleri ürettiğini ve UPMSPSDT çözümünde etkinliğini ortaya koyduğunu göstermektedir.

Anahtar kelimeler: Değişken Komşuluk Arama, Metasezgisel, Sıra bağımlı hazırlık süreli ilişkisiz paralel makine çizelgeleme.

1 Introduction

Scheduling process stands as a crucial process inherent in numerous production problems, wielding a substantial impact on the overall efficiency of the production process. Consequently, it demands meticulous planning by businesses to ensure optimal operational outcomes. Indeed, the intricacies of scheduling can differ significantly from one business to another as different production systems in workshops have caused distinct scheduling problems.

The aim of scheduling is to optimize one or more objectives while efficiently allocating resources to tasks. In the context of workshop scheduling, where resources typically refer to machines, and the tasks correspond to jobs. Scheduling problems can be classified according to the way jobs arrive or the number of machines. In deterministic scheduling problems, all job-related parameters such as processing times and setup times are known in advance before the scheduling begins. There is no change in the number of jobs or machines during scheduling [1]. When classifying the scheduling problems based on the number of machines involved, they can be broadly

*Corresponding author/Yazışılan Yazar

classified into two models: the single-machine and the multiple-machine models. In the case of scheduling with more than one machine, there are three basic models: parallel, serial and flow shop models [2]. In the context of parallel machine scheduling, unrelated machines refer to a set of machines that perform the same function but differ in their capabilities and capacities. An appropriate number of machines with different capabilities and capacities is widely used in many sectors such as textile, chemistry, electronic manufacturing, etc. As it increases flexibility in business [3]. In the Unrelated Parallel Machine Scheduling Problem with Sequence-Dependent Setup Times (UPMSPSDT), the processing times of each job on each machine are different. In this problem, the machines process the jobs with different capacities according to the jobs. At the same time, each job needs a setup time in that machine before it can be processed. The jobs have a setup time in the machine according to the last job processed before it. Even if there are no prior jobs distinct setup times may exist for each job on each machine.

The problem addressed in this study (UPMSPSDT), there are parallel machines and where the parallel machines are unrelated. The processing time of jobs on the machines are different. Additionally, setup times are both sequence-dependent, meaning they vary based on the preceding job, and machine-dependent. The identical parallel machine scheduling problem, which is a simpler version of this problem, is an Np-hard problem when the number of machines is 2 with no set up time [4]. As the number of jobs and machines increases, many practical scheduling problems exhibit growing complexity. Due to the NP-Hard nature of these problems, obtaining an optimal solution within a reasonable computational time may not always be feasible [5]. Given that a simpler version of this problem is NP-hard, it directly follows that the more complex UPMSPSDT is also NP-hard. In this problem, the objective is to minimize the final completion time (Cmax) from scheduling. Exact solution methods are not sufficient to solve the problems in the NP-hard class. Metaheuristic methods are used to solve this class of problems. Metaheuristics aim to iterate a candidate solution and improve the candidate solution according to the given criteria. Although they do not guarantee an exact solution, metaheuristic algorithms can solve optimization problems that other exact methods techniques cannot solve effectively [6].

The goal of this study is to propose an adapted VNS algorithm with a new proposed neighborhood structure (CNVNS), a new proposed greedy initial solution and a local search strategy and to find more efficient solution to the problem with this algorithm.

In this context, the past studies compared with the proposed algorithm are mentioned in the" Literature review" section. The commonly used neighborhood structures in VNS algorithm and the newly proposed neighborhood structure inspired by the "Crabs Shell Exchange" are given in the" Neighborhood structures" section. The modified version of greedy initial solution algorithm which is developed by [7] is given under the section" Proposed new initial solution". In this study, two local searches are also proposed to improve the result obtained from neighborhood exchange. The proposed local searches are evaluated and applied with a local search strategy according to the obtained Cmax values after neighborhood change.

2 Related literature

In this study, the benchmark dataset created by [6] was used to measure the effectiveness of the proposed metaheuristic algorithm. The problem and dataset of interest have been studied in academic research and many researchers have proposed solutions to the problem with different metaheuristics [6]-[18]. The benchmark dataset used and the results of other researchers can be found and a summary of the studies using the same benchmark dataset is given in Table 1.

As Table 1 illustrates, the UPMSPSDT has garnered significant attention from the research community, leading to the development of various metaheuristic algorithms. This table also motivates researchers to propose new methods and solutions for solving the problem.

UPMSPST has also been a focus of academic research, with various objective functions and datasets being explored. Some of the recent studies in this area include:

In the study by [19] a novel model for the UPMSPST was introduced, incorporating both limited workforce and the learning effects of workers. To solve this model, a Combinatorial Evolutionary Algorithm (CEA) was developed, which utilized List Scheduling (LS), the Shortest Setup Time (SST) priority rule, and the Earliest Completion Time (ECT) priority rule. For evaluation purposes, 72 test instances were generated, and the Taguchi method was applied to determine the optimal parameter combinations. The performance of the proposed algorithm was compared against other algorithms, demonstrating its effectiveness.

Table 1. A summary of the studies using the same benchmark dataset.

No	Reference	Algorithm	Compared
1	[6]	PH (Partitioning Heuristic)	-
2	[7]	Tabu Search	1
3	[8]	Meta-RaPS	1
4	[9]	Ant Colony Optimisation	1,2,3
5	[10]	Simulated Annealing and Genetic Algorithm	1
6	[11]	Restricted Simulated Annealing	1,2,3,4
7	[12]	Multi-start Variable Neighbourhood Descent	3,4
8	[13]	Artificial Bee Colony- Hybrid Artificial Bee Colony	1,2,3,4,5
9	[14]	Ant Colony Optimisation	1,2,3,4
10	[15]	Genetic Algorithm and Local Search	4,5
11	[16]	Automata- Adaptive Large Neighbourhood Search	4,9
12	[17]	Genetic Algorithm + Variable Neighbourhood Descent + Simulated Annealing (GIVP)	4,5,10
13	[18]	Worm Optimisation Algorithm	2,6,8,9,10

In the study by [20], the researchers addressed a real-world industrial problem in a textile factory, enhancing the parallel machine scheduling problem which has sequence-dependent setup times by incorporating two additional features: machine eligibility and limited resources. The goal of the study was to balance the workload by accounting for the dynamic layout between jobs in the workshop and identifying machine groups. A mathematical model was developed to represent the problem, and a genetic algorithm was employed to obtain solutions.

In the study by [21], the authors tackled the unrelated parallel machine energy-efficient scheduling problem with sequence-dependent setup times, considering varying energy consumption tariffs. The study examined setup times in two distinct ways: separately from processing times and jointly with processing times. For both cases, mixed-integer linear programming (MILP) models were developed. In the scenario where setup times were treated separately, solutions were achieved for problems involving up to 16 machines and 45 jobs, while for the joint setup time models, solutions were obtained for up to 20 machines and 40 jobs. To handle large-scale problems, the researchers also developed a fix-and-relax heuristic, allowing solutions for instances ranging from 20 to 100 jobs.

In their study, [22] focus on a multi-objective optimization problem in the context of the unrelated parallel machine scheduling problem, the objective is to minimize both the total tardiness and total completion time. To address this, they propose a novel metaheuristic algorithm as an alternative to the Augmented ϵ-Constraint (AEC) algorithm. The authors emphasize that their proposed algorithm demonstrates superior performance compared to the AEC algorithm, particularly in terms of solution quality and computational efficiency.

In the study by [23], the researchers aimed to solve same problem with a Group Genetic Algorithm, which is an enhanced version of the standard Genetic Algorithm. They introduced a novel mutation operator, referred to as ”2-Items,” resulting in a 52% improvement over the original mutation operator.

In another study, [24] tackled an unrelated parallel machine scheduling problem specific to Vestel. The goal was to minimize both early and tardy completion times in a production environment with unrelated parallel machines. To solve this, they employed the Imperial Competitive Algorithm. Their results demonstrated a 39% improvement over the current state-of-the-art methods, as well as surpassing the best algorithms in the literature by 23% and 12%.

In their study, [25] proposed a “fix and optimize” algorithm for the unrelated parallel machine scheduling problem, focusing on solving subproblems grouped by subsets of machines. To address these subproblems, they introduced a mathematical program known as MPA, which was used to obtain solutions. Computational experiments revealed that this approach produced significantly higher-quality solutions compared to a standalone exact algorithm. When compared to the best heuristic approaches in the literature, the proposed algorithm performed better in instances with a high number of jobs per machine but showed lower performance in instances with fewer jobs per machine.

In their 2024 study, [26] introduced an unrelated parallel machine scheduling problem inspired by a station in the wind tower production process, which involved setup times for the machines. They also incorporated a new feature, involving supportive machines that assist those facing capacity shortages. A mixed-integer linear programming (MILP) model was formulated for this problem, and solutions were explored using three different algorithms: a constructive heuristic, Simulated Annealing, and Tabu Search. The effectiveness of these algorithms was compared based on their performance in solving the problem.

This study [27] addresses a complex real-world scheduling problem that can be formulated as UPMSPSDT with due dates, and machine eligibility constraints. The main objective is to simultaneously minimize total tardiness and makespan. To achieve this, a mathematical model is adapted and extended to obtain optimal solutions for small-scale instances. For large-scale cases, various simulated annealing variants are proposed, leveraging different neighborhood structures and investigating innovative heuristic move selection strategies. The experimental results demonstrate that the methods are capable of improving upon the results produced by state-of-the-art approaches for a wide range of instances, including those drawn from the literature.

In the subsequent section, we’ll begin by presenting the newly proposed initial solution, essential for metaheuristic algorithms to commence their optimization process.

In the study [28], the authors introduced a Constraint Programming (CP) model designed to address UPMSPSDT. A key feature of their model is the integration of realistic constraints, including precedence relationships, release dates, and machine eligibility restrictions. To effectively solve this complex problem, they proposed two distinct branching strategies, which proved instrumental in successfully resolving small- and medium-sized problem instances. This work highlights the efficacy of CP in handling intricate scheduling challenges with practical relevance.

This study [29] investigates a variant of the Unrelated Parallel Machine Scheduling Problem with Sequence-Dependent Setup Times (UPMSPSDT), augmented with an additional resource constraint. The objective function is to minimize total weighted tardiness. In this context, a relaxed Mixed-Integer Linear Programming (MILP) model is proposed, integrated with a Constraint Programming (CP) component. For solving the problem, one heuristic (ATCS) and two meta-heuristic algorithms (GA and SA) were employed. The results were comparatively analyzed using both a raw benchmark dataset and a real-world problem instance.

In their study [30], beyond minimizing solely the makespan (Cmax), the authors aimed to schedule resource-constrained maintenance activities concurrently. To this end, a mixed-integer linear programming (MILP) model was proposed. For solving the real-world problem instance, which utilized actual factory data, a hybrid Genetic Algorithm (GA) model was developed. This model demonstrated superior results compared to the current factory practices.

This study [31] addresses an Unrelated Parallel Machine Scheduling Problem with Sequence-Dependent Setup Times (UPMSPSDT), uniquely extended to incorporate a personnel availability constraint for machine setups. Given that the number of available personnel is less than the number of machines, the scheduling must also account for personnel shift changes. Consequently, a Mixed-Integer Linear Programming

(MILP) model is formulated with the objective of minimizing Cmax (makespan). The practical utility of the proposed model has been demonstrated through its application to a real industrial case study.

In their study [32], the authors utilized the same dataset as the present work. They proposed a solution based on a Genetic Algorithm (GA) and compared its performance with that of the ACOII algorithm. It was reported that the GA demonstrated superior effectiveness, particularly in problems where setup times or processing times are dominant.

The problem addressed in this study [33] is the scheduling of jobs on unrelated parallel machines, considering both setup times and delivery times. In addition to proposing a mixed-integer programming model solved via CPLEX, a novel local search-based Adaptive Large Neighborhood Search (ALNS) algorithm has been developed to handle large-scale problem instances. To evaluate the performance of the proposed algorithm, a column generation approach is introduced, and comprehensive computational experiments are conducted on 4200 benchmark instances with up to 20 machines and 320 jobs. The results indicate that the proposed algorithm outperforms four state-of-the-art algorithms previously developed to solve similar problems.

3 Mathematical model

In addition to the heuristic solutions developed for the solution of the UPMSPSDST problem, an integer programming model is also presented [7]. The steps and formulas of the presented model are given below.

$$Minimize \quad C_{max} \tag{1}$$

Subject to

$$\sum_{\substack{i=0 \\ i\neq j}}^{n}\sum_{k=1}^{m} X_{i,j,k} = 1 \qquad \forall j = 1,\dots,n \tag{2}$$

$$\sum_{\substack{i=0 \\ i\neq h}}^{n} X_{i,j,k} - \sum_{\substack{j=0 \\ j\neq h}}^{n} x_{h,j,k} = 0 \qquad \forall h = 1,\dots,n, \forall k = 1,\dots,m \tag{3}$$

$$C_j \geq C_i + \sum_{k=1}^{m} X_{h,j,k}\left(S_{i,j,k} + P_{j,k}\right) + HV\left(\sum_{k=1}^{m} X_{i,j,k} - 1\right) \quad \forall h = 1,\dots,n, \forall i = 0,\dots,n, \forall j = 1,\dots,n \tag{4}$$

$$C_j \leq C_{max} \quad \forall j = 1,\dots,n \tag{5}$$

$$\sum_{j=0}^{n} X_{0,j,k} = 1 \quad \forall k = 1,\dots,m \tag{6}$$

$$C_j \geq 0 \quad \forall j = 1,\dots,n \tag{7}$$

$$C_0 = 0 \tag{8}$$

$$X_{i,j,k} \in \{0,1\} \quad \forall i = 0,\dots,n, \forall j = 0,\dots,n, \quad \forall k = 1,\dots,m \tag{9}$$

Where,

n: Number of jobs

m: Number of machines

C_j: Completion time of job j

$P_{j,k}$: processing time of job j on machine k

$S_{i,j,k}$: Setup time if job j is scheduled directly after job i on machine k

$S_{0,j,k}$: Setup time if job j is scheduled to go first on machine k

$X_{i,j,k}$: 1 if job j is scheduled directly after job i on machine k and 0 otherwise

$X_{0,j,k}$: 1 if job j is scheduled first on machine k and 0 otherwise

$X_{j,0,k}$: 1 if job j is scheduled last on machine k and 0 otherwise

HV: a large positive integer

The objective of the model, as defined in Equation (1), is to minimize the makespan (Cmax). Constraints (2) ensure that each job is assigned to only one machine and scheduled only once. Constraints (3) guarantee that each job has a predecessor and a successor. Constraints (4) serve a dual purpose. They calculate the completion time of each job (C_j) based on its predecessor (C_i) and the corresponding setup and processing times. Crucially, they also act as sub-tour elimination constraints based on the Miller-Tucker-Zemlin (MTZ) formulation. By enforcing a monotonically increasing sequence of completion times, these constraints make any cyclical assignment mathematically infeasible. Constraints (5) define the makespan by ensuring that it is no smaller than the completion time of any job. Constraint (6) ensures that each machine begins its sequence with exactly one job, which follows dummy job 0, thereby establishing a unique starting point for each machine. As a result, there is no need to explicitly define a constraint for the last job on each machine, since this condition is implicitly satisfied by Constraints (3) and (6). Constraints (7) and (8) ensure that completion times are non-negative and that the dummy job's completion time is zero, respectively. Lastly, constraints (9) state that the decision variable x is binary across all domains. Additionally, the model does not require explicit subtour elimination constraints (e.g., position-based variables like U_j because the completion time constraints (Constraints 4) inherently prevent cycles.

4 Proposed initial solution

There are two different approaches for generating an initial solution of metaheuristic algorithms: the generation of random initial solutions or the application of greedy algorithms. While greedy algorithms generate higher quality solutions, they increase the probability that the solution will be searched in a more restricted space and stuck at the local optimum. Random initialization algorithms, on the other hand, allow the heuristic algorithm to search more different areas in the solution space. In this case, finding the final solution is more difficult and will take longer time. The greedy algorithm proposed in this study is inspired and modified from the algorithm proposed in [7]. In the algorithm proposed by [7] the average of the setup times and the processing times are evaluated, and the jobs are assigned to the machine based on the ratio between the job with the smallest time and the next smallest time. If the calculated ratio is greater than the specified ratio, it adds the jobs to the list of unassigned jobs. In the second stage, it assigns the unassigned jobs above a certain ratio to the machine with the least completion time among the machines. In our proposed algorithm, diverging from the approach outlined in [7], the objective is to maintain balanced machine loads by consistently assigning jobs to the least loaded machine. Additionally, the focus lies on minimizing the overall completion time by strategically determining the best position for newly assigned jobs within the machine. The initial solution was derived from a modified version of an algorithm previously validated in a peer-reviewed scientific publication. Since it consistently outperforms random initial solutions, it was decided to conduct the actual experiments using this greedy initial solution rather

than random ones. The proposed initial solution algorithm is given in Algorithm 1.

Algorithm 1. The Proposed Greedy Initial Solution Function

Input: M: All machines, **N**: Set of jobs, **UM**: Unassigned machines.
Output: X: Final schedule of jobs assigned to machines
1: **procedure** GreedyInitialSolution (*M,N,UM*)
2: UM ←Ø, m ←Ø, j ←Ø, X ←Ø,
3: Calculate all avarage setup times for each machine
4: **for** *i* = 1 to N **do**
5: *m*=find least loaded machine
6: *j*= Find the job *j* with the smallest processing time + smallest average setup time on machine *m*
7: find optimal position for job *j* on machine *m*
8: X.append(*m*, *j*)
9: N.remove(*j*)
10: **end for**
11: **Return** X
12: **end procedure**

5 Neighborhood structures

To find a better solution for UPMSPSDT, typically 5 different neighborhood structures are used in the literature. Three of these are intra-machine job exchanges and two are inter-machine job exchanges. Intra-machine exchanges can be performed by swapping two jobs on the same machine, adding a job after a later job, or reversing the order between two jobs on a machine. Inter-machine job swapping can be done by swapping jobs on two different machines or by inserting a job on one machine behind a job on another machine [34]. In our study, alongside these five different neighborhood structures, a novel nature-inspired neighborhood exchange algorithm is introduced.

5.1 Proposed crab shell exchange algorithm

The vacancy chain in resource planning has also been described with crabs of the species Pagurus longicarpus. In resource allocation via vacancy chains, when an individual occupies an initially available resource unit, they vacate their previous unit, which becomes available for another person to take, and so forth. This process can be described as an interdependent sequence of consecutively vacated resource units. Such a chain facilitates the acquisition of new resource units by multiple individuals, contingent upon the prior resource acquisitions of others [35]. Crab shell exchange is modelled in two different ways: synchronous and asynchronous. Synchronized cavity chains occur when multiple crabs, positioned adjacent to an available empty shell, align in a descending order of size. Once the largest crab occupies the vacant shell, a swift sequence of successive shell exchanges follows. In asynchronous cavity chains, a crab that finds an appropriate empty shell will exchange its own shell and subsequently wait for another crab to discover and utilize the vacated shell. As a result, asynchronous gap chains are characterized by the absence of social interactions or queue formation, with sequential shell switching occurring over extended periods. In contrast, synchronized shell switching is more likely to result in the discovery of the most optimal shell [36].

The crabs' shell exchange algorithm was first proposed in 2019. The proposed algorithm is based on the search for a better shell than the shell acquired by each crab. The algorithm consists of 3 steps. Each crab finds a new shell randomly each time. The most suitable shell identified is carried over to the subsequent generation. The total number of shells remains constant, and the likelihood of crabs locating a shell is determined by predefined probabilities [37]. This method is similar to the asynchronous shell change of crabs.

Algorithm 2. Crab Neighbors Function

Input: M: number of machines, **N:** number of jobs, **X:** the initial schedule of jobs assigned to machines [[m_i, j_k], ...,] i: 1, ... , M, k:1, ... ,N, **UJ** : unassigned jobs
Output: X: the final schedule of jobs assigned to machines [[m_i, j_k], ...,]
1: **procedure** CrabNeighbors (*X, M, N, UJ*)
2: UJ ←Ø
3: **for** *i* = 1 to M **do**
4: *job*← find random job assigned to machine m_i in X // *each job is like a crab coming out of its shell*
5: UJ.append (job)
6: X. remove ([m_i, job])
7: **end for**
8: Machines =sort machines by workload descending in X // *like a chain of crabs*
9: **for each** mac in Machines **do** find job in UJ has least processing time in mac *// finding optimal Shell fort he crab*
11: X.append(mac, job)
12: UJ.remove(job)
13: **end for**
14: **Return** X
15: **end procedure**

In the proposed new neighborhood structure algorithm, the jobs selected from the machines are likened to crabs emerging from their shells and searching for new shells. The vacancies on the appropriate machine are calculated according to the loads on the remaining machines after removing one job from each machine. The pseudo code of this neighborhood structure is given in Algorithm 2. This algorithm is similar to synchronized shell exchange.

6 Proposed local search selection strategy and local search algorithms

Local search stands as a pivotal step aimed at enhancing an existing candidate solution. In VNS, local search is applied after neighborhood change to improve the initial solution.

In this study, the objective function is to minimize Cmax. For this minimization, 3 different local searches are proposed. Two of the proposed local searches are largescale and costly local searches. The first of these local searches LS1 (Replace Jobs on Critical Machine with All Jobs One by One) aims to replace all jobs on the critical machine with all jobs in the list and to obtain a more appropriate solution value according to the objective function. The second local search LS2 (Insert Jobs on Critical Machine with All Possibilities One by One) aims to obtain the more appropriate solution value by inserting all the jobs on the critical machine before the jobs on other machines. If the job added before is the last job to be processed on the machine, it also tries to add it after it. LS1 and LS2 local searches work effectively if there is only 1 machine that is costlier than the best solution found in the objective function. If there is more than

one machine that is costlier than the critical machine according to the previous iteration, changing the jobs on the critical machine will not allow improvement in the objective function.

LS1 and LS2 local search only deals with the jobs on the critical machine. The third local search LS3 (Change Two Critical Machines Jobs) deals with 2 critical machines at the same time, not with jobs on a single machine like LS1 and LS2. LS3 aims at better scheduling of both machines at the same time by replacing all the jobs on 2 machines that are larger than the Cmax value obtained in the previous iteration.

The order of use of local searches is linked to a selection strategy based on the best known Cmax value and the machine loads obtained by neighborhood change. In this context, after each neighborhood exchange, the workloads (C values) of the machines are compared with the best known Cmax value. If there is 1 critical machine with a value greater than Cmax or if the Cmax value has decreased with the neighborhood change, LS1 and LS2 local searches are used to decrease the workload of this machine. In other words, if only one machine exceeds the current Cmax after a neighborhood move, focusing the local search on that specific machine—through LS1 and LS2—allows for a more targeted and effective reduction of the makespan. Since other local searches involve multiple machines, they may not yield meaningful improvements in such cases. Thus, this selective application ensures a more efficient use of local search operators. If there is more than one critical machine, these machines are handled together. The local search selection strategy using these local searches is given in Algorithm 3.

Algorithm 3 Local Search Choose Function

Input: *Cmaxbest***:** Stored best *Cmax* value, **X'**: Neighborhood applied list **LS1**: Replace Jobs on Critical Machine with All Jobs One by One, **LS2**= Insert Jobs on Critical Machine with All Possibilities One by One, **LS3**= Change Two Critical Machines Jobs, cmcount= Critical machine count
Output: X: Final schedule of jobs assigned to machines

```
1: procedure Local Search Choose (X', LS1, LS2, LS3, Cmaxbest)
2:     cmcount ← 0
3:     Calculate each machine Cvalues of (X')
4:     cmcount =compare (Cvalues, Cmaxbest)
5:     if cmcount <2 then
6:         X" = Apply LS1 to X'
7:         if Cmax of X" <Cmaxbest then
8:             X←X", X'←X", Cmaxbest ← Cmax of X"
9:         end if
10:        X" = Apply LS2 to X'
11:        if Cmax of X" <Cmaxbest then
12:            X←X", X'←X", Cmaxbest ← Cmax of X"
13:    end if
14:  else
15:        X" = apply LS3 to X'
16:        if Cmax of X" <Cmaxbest then
17:            X←X", X'←X", Cmaxbest ← Cmax of X"
18:        end if
19:  end if
20: Return X
21: end procedure
```

As seen in Algorithm 3, local search selection is evaluated according to Cmax values. LS1, LS2, LS3 local search algorithms used in Algorithm 3 are given respectively. The first of these, LS1, is given in Algorithm 4.

Algorithm 4 Replace Jobs on Critical Machine with All Jobs One by One.

Input: X': Neighborhood applied list **M:** All machines, *j***:** Selected job, $\mathbf{m_k}$**:** Critical machine, $\mathbf{j_m}$**:** Job on machine m, X=stored best machine and job configurations **Output: X:** Final schedule of jobs assigned to machines

```
1: procedure Replace Jobs on Critical Machine with All Jobs One by One(X, j, mk, jm)
2:     mk = find critical machine ()
3:     for each j ϵ mk do
4:         for each m ϵ M do
5:             for each jm ϵ m do
6:                 X" =Swap (j, jm)
7:                 Cmaxnew=calculateCmax(X")
8:                 if Cmaxnew <Cmaxbest then
9:                     X←X"
10:                end if
11:            end for
12:        end for
13:    end for
14: Return X
15: end procedure
```

Algorithm 4 is a simple algorithm. The processing time increases according to the number of machines and jobs. The computational complexity of LS1 algorithm is calculated as follows. N is the total number of jobs, M is the total number of machines, and if it is assumed that the jobs are evenly distributed to the machines, each machine is expected to have N / M number of jobs. In LS1, since each job on the critical machine is replaced by all other jobs, the computational complexity is N * (N / M). Another local search LS2 is given in Algorithm 5.

Algorithm 5 Insert Jobs on Critical Machine with All Possibilities One by One.

Input: X': Neighborhood applied list **M:** All machines, *j*: selected job, m_k: Critical machine, j_m: job on machine *m*,
Output: X: Final schedule of jobs assigned to machines

```
1: procedure Insert Jobs on Critical Machine with All Possibilities One by One (X, j, mk, jm)
2:     mk = find critical machine ()
3:     for each j ϵ mk do
4:     for each m ϵ M do
5:         for each jm ϵ m do
6:             X"←Insert (j before jm)
7:             Cmaxnew=calculateCmax(X")
8:             if Cmaxnew<Cmaxbest then
9:                 X←X"
10:            end if
11:            if jm is last job on machine then
12:                X"←Insert (j after jm)
13:                Cmaxnew=calculateCmax(X")
14:            end if
15:            if Cmaxnew < Cmaxbest then
16:                X←X"
17:            end if
18:        end for
19:    end for
```

20: **end for**
21: **Return** X
22: **end procedure**

In LS2, jobs can be added before and after the jobs on the machine. With N being the total number of jobs and M being the total number of machines, the jobs assigned to the critical machine can be added to N + M places. In this case, the computational complexity is (N + M) * (N / M).

Algorithm 5 works in the same way as Algorithm 4 and searches for a better Cmax value by adding each job on the critical machine to each location on all machines. As mentioned before, when there is more than one critical machine with a completion time greater than Cmaxbest due to neighborhood change, performing an operation on one of these machines will not affect the other critical machine, so there will be no decrease in Cmax. For this reason, in this case, a better solution is searched with the LS3 algorithm in which only these two machines are considered. LS3 algorithm is given in Algorithm 6.

Algorithm 6 Change two critical machines jobs.

Input: X': Neighborhood applied list **M:** All machines, *j*: selected job, m_{k1}: Critical machine1, m_{k2}: Critical machine2, j_{m1}: job on machine m_{k1}, j_{m2}: job on machine m_{k2}, *Cmaxbest*: Best *Cmax* value reached so far.
Output: X=Stored best machine and job configurations
1: **procedure** Change two critical machines jobs (X, *j*, m_k, j_m)
2: m_{k1}= find critical machine1
3: m_{k2}= find critical machine2
4: **for each** $j_{m1} \epsilon m_{k1}$ **do**
5: **for each** $j_{m2} \epsilon m_{k2}$ **do**
6: **X"←swap** (j_{m1}, j_{m2})
7: *Cmaxnew*=calculateCmax(**X"**)
8: **if** *Cmaxnew < Cmaxbest* **then**
9: **X←X"**
10: **end if**
11: **end for**
12: **end for**
13: **Return** X
14: **end procedure**

Algorithm 6 seeks an opportunity if there is a chance in changing jobs on both critical machines which has more Cmax values. In LS3, the computational complexity is (N / M) * (N / M) because only jobs on two machines are interchanged.

7 Proposed CNVNS algorithm

The VNS algorithm, a modern heuristic method, was developed by Nenad Mladenovic and Pierre Hansen in 1997. Since its inception, the algorithm has been continuously refined and widely applied across various domains. VNS is a metaheuristic solution approach that simultaneously utilizes different neighborhood structures. Based on the principle of systematically altering the neighborhood structures during the search process, DKA is a simple, yet effective heuristic designed to solve combinatorial and global optimization problems [38]. Although there are many variants of VNS algorithms [39], this paper presents an adapted version of the basic VNS. The new proposed algorithm uses the proposed initial solution, neighborhood structures and local search techniques. The Basic VNS algorithm is given in Algorithm 7.

Algorithm 7 Basic VNS Algorithm

Input: X: Initial Solution N_k, k=1,2, ..., *kmax*: Neighborhood structures, LS: Local Search, f(.): objective function
Output: X: Final schedule of jobs assigned to machines
1: **procedure** VNS
2: **while** the stop condition is not met **do**
3: k=1
4: **while** k <= kmax **do**
5: X'← N_k(X)
6: X"← LS(X')
7: **if** f(X")<f(X) **then**
8: k←1, X← X"
9: **else**
10: k←k+1
11: **end if**
12: **end while**
13: **end while**
14:**Return** X
15:**end procedure**

The new proposed algorithm uses the proposed initial solution, neighborhood structures and local search techniques. The Basic VNS algorithm is given in Algorithm 7 and the proposed CNVNS (Crab Neighborhood + VNS) algorithm is given in Algorithm 8.

The basic VNS algorithm is modified as follows according to proposed algorithms in this paper and given in Algorithm 8. The proposed CNVNS algorithm differs from the basic VNS algorithm in several key aspects, including the method of initial solution generation, the design of neighborhood structures, and the selection of local search strategies.

Algorithm 8 Proposed CNVNS Algorithm.

Input: X: Initial Solution from Algorithm 1: GreedyInitialSolution Nk: k=1,2, ..., kmax: neighborhood structures LS: Local Search phase (Algorithm 3: LocalSearchChoose) which includes Algoritm), f(.): objective function Output: X: Final schedule of jobs assigned to machines
1: **procedure** CNVNS
2: **while** the stop conditon is not met **do**
3: k=1
4: **while** k <=kmax **do**
5: X'←N_k(X) *// CrabNeighbors neighborhood structure added to* N_k
6: X"← LS (X') *// LocalSearchChoose algorithm which uses Algorithm* 4, *Algorithm* 5 *and Algorithm* 6 *systematically*
7: **if** f(X")<f(X) **then**
8: k←1, X← X"
9: **else**
10: k←k+1
11: **end if**
12: **end while**
13: **end while**
14: **Return** X
15: **end procedure**

8 Computational experiments and discussion

8.1 Benchmark dataset

The benchmark dataset in this study was developed by [6] and has been used by many researchers mentioned in Table 1. The dataset is categorized into two groups: small-size problems and large-size problems. The small-size dataset utilized in this study comprises 270 distinct data files that need to be scheduled on machines with different numbers of jobs j ∈ {6, 7, 8, 9, 10 and 11} and different numbers m ∈ {2, 4, 6 and 8}. The large data set consists of 540 data files in which j ∈ {20, 40, 60, 80, 100 and 120} jobs are scheduled on m ∈ {2, 4, 6, 8, 10 and 12} machines with balanced setup and processing time. In the large dataset, there are a total of 1080 data files which setup time is dominant, and processing time is dominant. This study is limited to 540 big data with balanced setup and processing time. Data and related solutions of existing algorithms are available on [40].

8.2 Testing CNVNS

The stopping criteria for the VNS algorithm can include the maximum execution time, the maximum number of iterations, or the maximum number of iterations without any improvement [39]. Different stopping criteria and algorithm execution times have been determined by researchers in the solution of the problem studied. [10] set the maximum number of iterations as 500 times the number of jobs [11] terminated the algorithm if there was no improvement for 214 iterations or 28 times. [7] tied intra-machine and inter-machine switching to an iteration number that varies with the number of machines and ran inter-machine switching at least 25 times. [9] stated that a small number of iterations is sufficient for relatively small problems, however, they ran the algorithm for 15000 iterations to obtain the best-known results on large data sets. As can be seen, there is no standard in the stopping criterion among researchers.

The software platform Pycharm 2022.1.2 was selected to develop the proposed algorithms and the algorithms were coded in Python (3.7). The configuration of the computer on which the study was run is (Intel(R) Core (TM) i7-6700 CPU @ 3.40GHz, 16GB Ram). The proposed algorithm was run one iteration for 540 instances on a balanced large dataset. Execution times are reported for all machine job variations. Average times for each machine and number of jobs are given in Table 2.

In Table 2, the average times show that number of jobs increases also the time increases as expected. When 540 samples are considered as a whole, the total running time for all samples in the benchmark dataset is 24297.374 s. This time is equal to 404.956 minutes and 6.75 hours. In case of an improvement in the neighborhood change or local search phase of the VNS, the neighborhood change is moved to the beginning, which causes the first iteration to be longer. When the total time of 24297.374 seconds is divided by the size of instances (540), the running time per instance is 44.995 seconds. This time for one iteration can be considered reasonable. As stated above, there is no standardized stopping criterion for the algorithm. A reasonable number of iterations was determined and tested to ensure adequate runtime for the algorithms. The algorithm was run for 100 iterations on small and large data sets and the test results are given in Table 3 and Table 4. In the given tables the best-known result of each machine job combination is indicated in bold and italic. If there are equal best solutions they are all indicated as bold and italic. Firstly, the success of the proposed algorithm is tested with small size problems in the benchmark dataset. The algorithm is run and the best results are reported. For 270 data sets, Cmax values are grouped according to the number of machines and jobs. The test results are shown in Table 3[17].

When analyzing Table 3, it is evident that the CNVNS algorithm achieves successful results on the small dataset. A direct comparison with previously applied algorithms on the same dataset reveals that the proposed algorithm consistently attains optimal or superior outcomes across numerous instances.

Table 2. Average execution times of the proposed algorithm for one iteration.

Machine	Job	Average Time (s)	Machine	Job	Average Time (s)
2	20	1.065	8	20	0.289
2	40	9.932	8	40	1.894
2	60	35.648	8	60	11.215
2	80	92.260	8	80	24.851
2	100	134.340	8	100	49.227
2	120	235.757	8	120	111.676
4	20	0.364	10	20	0.083
4	40	5.603	10	40	1.144
4	60	21.956	10	60	6.336
4	80	63.842	10	80	19.354
4	100	118.003	10	100	36.384
4	120	176.802	10	120	78.832
6	20	0.381	12	20	0.093
6	40	2.735	12	40	1.406
6	60	15.595	12	60	4.417
6	80	44.827	12	80	10.376
6	100	82.566	12	100	39.843
6	120	129.759	12	120	50.968

Table 3. Minimum Cmax value averages of small data set.

Mach.-Job	[13]	[8]	[7]	[6]	CNVNS	Optimal
2-6	**394.73**	**394.73**	**394.73**	396.40	**394.73**	**394.73**
2-7	**491.00**	**491.00**	**491.00**	495.07	**491.00**	**491.00**
2-8	**517.40**	**517.40**	**517.40**	522.60	**517.40**	**517.40**
2-9	**598.47**	**598.47**	598.87	603.80	**598.47**	**598.47**
2-10	**638.93**	**638.93**	**638.93**	645.33	**638.93**	
2-11	710.73	710.73	710.73	721.27	**710.40**	
4-6	**245.00**	**245.00**	**245.00**	251.73	**245.00**	**245.00**
4-7	**252.27**	**252.27**	**252.27**	265.07	**252.27**	**252.27**
4-8	**264.73**	**264.73**	**264.73**	271.27	**264.73**	**264.73**
4-9	**346.07**	**346.07**	346.87	346.40	**346.07**	
4-10	**359.47**	359.53	**359.47**	361.60	**359.47**	
4-11	**366.33**	366.47	**366.33**	374.33	**366.33**	
6-8	234.47	234.47	234.47	242.07	**234.47**	
6-9	238.53	238.53	238.53	249.53	**238.53**	
6-10	246.00	246.47	246.00	259.47	**245.93**	
6-11	251.27	251.60	251.27	274.53	**250.80**	
8-10	**226.13**	**226.13**	**226.13**	232.00	**226.13**	
8-11	**232.47**	**232.47**	**232.47**	235.60	**232.47**	

Table 4. Min Cmax value averages in balanced-large data set.

Machine-Job	[17]	[15]	[13]	[13]	[11]	[9]	[7]	CNVNS
2-20	1240.27	1236.20	1234.87	1235.80	1238.53	1237.80	1264.87	***1234.87***
2-40	2436.80	2425.87	2400.27	2405.33	2411.27	***2397.80***	2486.53	2400.60
2-60	3653.40	3641.60	3583.80	3591.87	3598.87	***3574.60***	3736.47	3583.73
2-80	4844.60	4834.00	4755.33	4764.13	4776.07	***4730.40***	4942.27	4753.60
2-100		6019.80	***5924.20***	5936.07	5937.20	5897.60	6180.87	5925.00
2-120		7232.80	7126.27	7135.53	7144.40	***7082.60***	7447.60	7124.60
4-20	610.07	609.73	608.33	609.40	609.80	617.13	622.93	***608.27***
4-40	1181.27	1182.73	1158.40	1161.67	1165.13	1179.87	1200.67	***1158.13***
4-60	1759.80	1757.00	1718.13	1725.20	1729.07	1737.93	1785.53	***1717.40***
4-80	2339.27	2337.87	2286.87	2294.40	2302.53	2298.53	2370.13	***2284.67***
4-100		2892.53	2835.13	2850.60	2852.47	2849.93	2934.13	***2834.93***
4-120		3465.13	3400.67	3413.40	3410.20	3405.13	3515.13	***3394.07***
6-20	446.20	445.93	***445.87***	446.00	446.13	452.73	449.40	***445.87***
6-40	794.73	790.87	779.40	782.93	783.67	805.40	803.73	***778.00***
6-60	1162.13	1162.40	1133.47	1137.67	1141.20	1163.47	1179.27	***1133.20***
6-80	1552.87	1543.93	1514.13	1523.07	1527.93	1545.33	1568.60	***1511.40***
6-100		1910.13	1869.33	1881.33	1883.47	1897.47	1940.60	***1867.73***
6-120		2286.13	2235.20	2241.60	2252.40	2253.93	2313.07	***2227.73***
8-20	342.07	340.13	***339.47***	339.87	339.73	347.60	342.80	***339.47***
8-40	585.47	589.27	***572.67***	578.53	577.60	599.27	588.67	573.27
8-60	888.33	882.67	867.93	873.93	874.80	893.80	893.13	***865.80***
8-80	1150.07	1150.20	***1116.27***	1122.80	1125.73	1142.40	1164.60	1116.73
8-100	1438.73	1430.87	1403.73	1411.87	1414.40	1439.07	1449.27	***1399.40***
8-120	1707.33	1713.07	1664.47	1675.73	1685.40	1686.07	1739.73	***1658.87***
10-20	248.20	245.73	***242.73***	245.93	244.13	252.53	260.20	***242.73***
10-40	470.60	474.53	***459.13***	465.93	464.20	485.53	474.60	459.20
10-60	695.27	699.47	***673.20***	682.13	682.27	708.27	692.73	674.27
10-80	920.53	926.60	***893.20***	902.33	901.53	925.87	920.80	893.27
10-100	1141.53	1149.00	1107.67	1117.00	1115.20	1141.53	1153.27	***1106.07***
10-120	1362.20	1378.60	1326.87	1334.33	1332.27	1351.67	1376.33	***1320.60***
12-20	233.67		***231.00***	232.00	231.20	241.87	245.00	***231.00***
12-40	437.87		431.13	433.67	433.47	448.13	436.87	***430.93***
12-60	581.07		***561.87***	572.20	570.33	597.33	576.87	563.67
12-80	780.80		762.27	768.73	766.27	790.07	778.47	***759.53***
12-100	978.93		961.40	963.07	964.27	988.67	981.73	***951.40***
12-120	1136.67		1105.87	1115.33	1110.27	1138.73	1146.40	***1099.60***

In metaheuristic algorithms, it is accepted if the gap is less than gap <0,1 according to the gap= (f − f0) % f value calculated in small size problems [41]. CNVNS algorithm has reached optimal values in all the problems which have optimal solutions in the small size data set. Therefore, the algorithm can be deemed effective based on these outcomes.

The CNVNS algorithm was also run 100 times on the Balanced-large data set. Cmax values for 540 data sets are grouped according to the number of machines and jobs. The results are given in Table4.

Analyzing Table 4 reveals that the proposed CNVNS algorithm outperforms competing metaheuristic algorithms, demonstrating a higher success rate. It achieved the best performance in 25 out of 36 machine-job configurations (69.44%) and matched or performed better result in 21 cases (58.2%). These results highlight the algorithm's effectiveness across various configurations. In the article [42], the authors emphasize that the comparative evaluation of optimization algorithms involves numerous methodological complexities. They highlight that, to ensure a fair and valid comparison, factors such as algorithm configuration, parameter settings, characteristics of test instances, and hardware conditions must be carefully controlled. Therefore, the algorithms have been compared solely based on their Cmax values.

9 Conclusions

In this study, a novel metaheuristic algorithm is proposed that introduces a novel approach to initial solutions through a newly devised greedy algorithm. This innovative greedy algorithm is specifically designed to expedite goal achievement while exploring the solution space, thereby ensuring a high-quality initial solution. In addition, a nature-inspired neighborhood structure simulating animal behavior is presented in this study. In the local search phase, the proposed three different local searches are efficiently used with a selection algorithm to avoid unnecessary moves. The algorithm tested on a benchmark problem, and it has been demonstrated that the proposed method yields superior results when compared to the alternative algorithms in the literature. As a further study, the proposed CNVNS algorithm can be combined with other algorithms to obtain new hybrid algorithms to achieve even more successful results.

This study provides actionable insights for production planners and operations managers. Minimizing the makespan enables faster order fulfillment, improved resource utilization, and higher production efficiency. The adaptive local search strategy dynamically identifies bottlenecks and selects appropriate operators, demonstrating the value of flexible optimization over rigid heuristics. Additionally, the use of a load-balanced greedy algorithm for generating initial solutions accelerates convergence and reduces computational effort, offering a practical advantage in real-world scheduling environments.

10 Author contributions statement

In this study, Günay Kılıç contributed to the formation of the idea, the design process, the literature review, and the obtaining of results. Arzu Organ contributed to the evaluation of the obtained results, proofreading, and the academic review of the content. This study is derived from the doctoral dissertation titled "A new metaheuristic proposal for unrelated parallel machine scheduling problem with sequence-dependent setup times (Sıra bağımlı ilişkisiz paralel makine çizelgeleme problemi için yeni bir sezgisel algoritma önerisi)" (No: 794919), conducted under the supervision of Prof. Dr. Arzu ORGAN

11 Ethics committee approval and conflict of interest statement

"There is no need to obtain an ethics committee approval for the article prepared". "There is no conflict of interest with any person/institution in the article prepared".

12 References

[1] Pinedo ML, *Scheduling: Theory, Algorithms, and Systems.* 5th ed. USA, Springer, 2016.

[2] Baker KR, Trietsch D. *Principles of Sequencing and Scheduling*. 2nd ed. USA, John Wiley & Sons, 2019.

[3] Lin SW, Lu CC, Ying KC. "Minimization of total tardiness on unrelated parallel machines with sequence-and machine-dependent setup times under due date constraints". *The International Journal of Advanced Manufacturing Technology*, 53, 353-361, 2011.

[4] Arnaout JP. "A worm optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times". *Annals of Operations Research*, 285(1), 273-293, 2020.

[5] Sel Ç, Hamzadayı A. "A simulated annealing approach-based simulation-optimisation to the dynamic job-shop scheduling problem". *Pamukkale Üniversitesi Mühendislik Bilimleri Dergisi*, 24(4), 665-674, 2018.

[6] Al-Salem A. "Scheduling to minimize makespan on unrelated parallel machines with sequence dependent setup times". *Engineering Journal of the University of Qatar,* 17(1), 177-187, 2004.

[7] Helal M, Rabadi G, Al-Salem A. "A tabu search algorithm to minimize the makespan for the unrelated parallel machines scheduling problem with setup times". *International Journal of Operations Research*, 3(3), 182-192, 2006.

[8] Rabadi G, Moraga RJ, Al-Salem A. "Heuristics for the unrelated parallel machine scheduling problem with setup times". *Journal of Intelligent Manufacturing*, 17(1), 85-97, 2006.

[9] Arnaout JP, Rabadi G, Musa R. "A two-stage ant colony optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times". *Journal of Intelligent Manufacturing*, 21(6), 693-701, 2010.

[10] Chang PC, Chen SH. "Integrating dominance properties with genetic algorithms for parallel machine scheduling problems with setup times". *Applied Soft Computing*, 11(1), 1263-1274, 2011.

[11] Ying KC, Lin SW. "Unrelated parallel machine scheduling with sequence-and machine-dependent setup times and due date constraints". *International Journal of Innovative Computing, Information and Control*, 8(5), 3279-3297, 2012.

[12] Fleszar K, Charalambous C, Hindi KS. "A variable neighborhood descent heuristic for the problem of makespan minimisation on unrelated parallel machines with setup times". *Journal of Intelligent Manufacturing*, 23(5), 1949-1958, 2012.

[13] Lin SW, Ying KC. "ABC-based manufacturing scheduling for unrelated parallel machines with machine-dependent and job sequence-dependent setup times". *Computers & Operations Research*, 51, 172-181, 2014.
[14] Arnaout JP, Musa R, Rabadi G. "A two-stage Ant Colony optimization algorithm to minimize the makespan on unrelated parallel machines-part II: enhancements and experimentations". *Journal of Intelligent Manufacturing*, 25(1), 43-53, 2014.
[15] Yilmaz Eroğlu D, Ozmutlu HC, Ozmutlu S. "Genetic algorithm with local search for the unrelated parallel machine scheduling problem with sequence-dependent set-up times". *International Journal of Production Research*, 52(19), 5841-5856, 2014.
[16] Cota LP, Guimarães FG, de Oliveira FB, Souza MJF. "An adaptive large neighborhood search with learning automata for the unrelated parallel machine scheduling problem". *IEEE Congress on Evolutionary Computation,* San Sebastián, Spain, 5-8 June 2017.
[17] De Abreu LR, de Athayde Prata B. "A genetic algorithm with neighborhood search procedures for unrelated parallel machine scheduling problem with sequence-dependent setup times". *Journal of Modelling in Management*, 15(3), 809-828, 2020.
[18] Arnaout JP. "A worm optimization algorithm to minimize the makespan on unrelated parallel machines with sequence-dependent setup times". *Annals of Operations Research*, 285(1), 273-293, 2020.
[19] Zhang L, Deng Q, Lin R, Gong G, Han W. "A combinatorial evolutionary algorithm for unrelated parallel machine scheduling problem with sequence and machine-dependent setup times, limited worker resources and learning effect". *Expert Systems with Applications*, 175, 114843, 2021.
[20] Berthier A, Yalaoui A, Chehade H, Yalaoui F, Amodeo L, Bouillot C. "Unrelated parallel machines scheduling with dependent setup times in textile industry". *Computers & Industrial Engineering,* 174, 108736, 2022.
[21] Sanati H, Moslehi G, Reisi-Nafchi M. "Unrelated parallel machine energy-efficient scheduling considering sequence-dependent setup times and time-of-use electricity tariffs". *EURO Journal on Computational Optimization*, 11, 100052, 2023.
[22] Sarac T, Ozcelik F. "A matheuristic algorithm for multi-objective unrelated parallel machine scheduling problem". *Journal of the Faculty of Engineering and Architecture of Gazi University*, 38(3), 1953-1966, 2023.
[23] Ramos-Figueroa, O, Quiroz-Castellanos M, Mezura-Montes E, Cruz-Ramirez N. "An experimental study of grouping mutation operators for the unrelated parallelmachine scheduling problem". *Mathematical and Computational Applications*, 28(1), 6, 2023.
[24] Elyasi M, Selcuk YS, Özener OÖ, Coban, E. "Imperialist competitive algorithm¨ for unrelated parallel machine scheduling with sequence-and-machine-dependent setups and compatibility and workload constraints". *Computers & Industrial Engineering*, 190, 110086, 2024.
[25] Fonseca GH, Figueiroa GB, Toffolo TA. "A fix-and-optimize heuristic for the unrelated parallel machine scheduling problem". *Computers & Operations Research*, 163, 106504, 2024.
[26] Munoz-Diaz ML, Escudero-Santana A, Lorenzo-Espejo A. "Solving an unrelated parallel machines scheduling problem with machine-and job-dependent setups and precedence constraints considering support machines". *Computers & Operations Research*, 163, 106511, 2024.
[27] Moser M, Musliu N, Schaerf A, Winter F. "Exact and metaheuristic approaches for unrelated parallel machine scheduling". *Journal of Scheduling*, 25(5), 507-534, 2022.
[28] Yunusoglu P, Topaloglu Yildiz S. "Constraint programming approach for multi-resource-constrained unrelated parallel machine scheduling problem with sequence-dependent setup times". *International Journal of Production Research*, 60(7), 2212-2229, 2022.
[29] Avgerinos I, Mourtos I, Vatikiotis S, Zois G. "Weighted tardiness minimisation for unrelated machines with sequence-dependent and resource-constrained setups". *International Journal of Production Research*, 62(1-2), 359-379, 2024.
[30] Geurtsen M, Adan J, Akçay A. "Integrated maintenance and production scheduling for unrelated parallel machines with setup times". *Flexible Services and Manufacturing Journal*, 36 ,1046-1079, 2023.
[31] Khadivi M, Abbasi M, Charter T, Najjaran H. "A mathematical model for simultaneous personnel shift planning and unrelated parallel machine scheduling". https://doi.org/10.48550/arXiv.2402.15670
[32] Antunes AR, Matos MA, Rocha AMA, Costa LA, Varela LR. "A statistical comparison of metaheuristics for unrelated parallel machine scheduling problems with setup times". *Mathematics*, 10(14), 2431, 2022.
[33] Xie F, Li K, Chen J, Xiao W, Zhou T. "An adaptive large neighborhood search for unrelated parallel machine scheduling with setup times and delivery times". *Computers & Operations Research*, 177, 106976, 2025.
[34] Wang L, Wang S, Zheng X. "A hybrid estimation of distribution algorithm for unrelated parallel machine scheduling with sequence-dependent setup times". *IEEE/CAA Journal of Automatica Sinica*, 3(3), 235-246, 2016.
[35] Chase ID, Weissburg M, Dewitt TH. "The vacancy chain process: a new mechanism of resource distribution in animals with application to hermit crabs". *Animal behaviour*, 36(5), 1265-1274, 1988.
[36] Rotjan RD, Chabot JR, Lewis SM, "Social context of shell acquisition in Coenobita clypeatus hermit crabs". *Behavioral Ecology*, 21(3), 639-646, 2010.
[37] Murali GB, Biswal BB, Deepak BBVL, Rout A, Mohanta GB. "A New Crab Shell Search Algorithm for Optimal Assembly Sequence Generation". *9th Annual Information Technology, Electromechanical Engineering and Microelectronics Conference*, United States, 13-15 March 2019.
[38] Topaloğlu D, Polat O, Kalaycı CB. "Çok kompartımanlı heterojen filolu zaman pencereli araç rotalama probleminin çözümü için sezgisel algoritmalar". *Pamukkale Üniversitesi Mühendislik Bilimleri Dergisi*, 29(8), 870-884, 2023.
[39] Hansen P, Mladenovic N. *Developments of Variable Neighborhood Search.* Editors: Ribeiro CC, Hansen P. Essays and Surveys in Metaheuristics, 415–439, Boston, MA, Springer, 2002.

[40] SVC. "Scheduling Virtual Center". http://schedulingresearch.com (01.09.2022).

[41] Jiang Z, Chen, Y, Li X, Li B. "A heuristic optimization approach for multi-vehicle and one-cargo green transportation scheduling in shipbuilding". *Advanced Engineering Informatics*, 49, 101306, 2021.

[42] Beiranvand V, Hare W, Lucet Y. "Best practices for comparing optimization algorithms". *Optimization and Engineering*, 18, 815-848, 2017.