Hybrid intelligent water drops algorithm to unrelated parallel machines scheduling problem: a just-in-time approach

Vahid Kayvanfar\textsuperscript{a} & Ehsan Teymourian\textsuperscript{b}

\textsuperscript{a} Department of Industrial Engineering, Amirkabir University of Technology, Tehran, Iran
\textsuperscript{b} Department of Industrial Engineering, Mazandaran University of Science and Technology, Babol, Iran

Published online: 04 Jun 2014.

To cite this article: Vahid Kayvanfar & Ehsan Teymourian (2014) Hybrid intelligent water drops algorithm to unrelated parallel machines scheduling problem: a just-in-time approach, International Journal of Production Research, 52:19, 5857-5879, DOI: 10.1080/00207543.2014.923124

To link to this article: http://dx.doi.org/10.1080/00207543.2014.923124

PLEASE SCROLL DOWN FOR ARTICLE

Taylor & Francis makes every effort to ensure the accuracy of all the information (the “Content”) contained in the publications on our platform. However, Taylor & Francis, our agents, and our licensors make no representations or warranties whatsoever as to the accuracy, completeness, or suitability for any purpose of the Content. Any opinions and views expressed in this publication are the opinions and views of the authors, and are not the views of or endorsed by Taylor & Francis. The accuracy of the Content should not be relied upon and should be independently verified with primary sources of information. Taylor and Francis shall not be liable for any losses, actions, claims, proceedings, demands, costs, expenses, damages, and other liabilities whatsoever or howsoever caused arising directly or indirectly in connection with, in relation to or arising out of the use of the Content.

This article may be used for research, teaching, and private study purposes. Any substantial or systematic reproduction, redistribution, reselling, loan, sub-licensing, systematic supply, or distribution in any form to anyone is expressly forbidden. Terms & Conditions of access and use can be found at http://www.tandfonline.com/page/terms-and-conditions
Hybrid intelligent water drops algorithm to unrelated parallel machines scheduling problem: a just-in-time approach

Vahid Kayvanfar* and Ehsan Teymourian

aDepartment of Industrial Engineering, Amirkabir University of Technology, Tehran, Iran; bDepartment of Industrial Engineering, Mazandaran University of Science and Technology, Babol, Iran

(Received 2 July 2013; accepted 4 May 2014)

Minimising earliness and tardiness penalties as well as maximum completion time (makespan) simultaneously on unrelated parallel machines is tackled in this research. Jobs are sequence-dependent set-up times and due dates are distinct. Since the machines are unrelated, jobs processing time/cost on different machines may vary, i.e. each job could be processed at different processing times with regard to other machines. A mathematical model which minimises the mentioned objective is proposed which is solved optimally via lingo in small-sized cases. An intelligent water drop (IWD) algorithm, as a new swarm-based nature-inspired optimisation one, is also adopted to solve this multi-criteria problem. The IDW algorithm is inspired from natural rivers. A set of good paths among plenty of possible paths could be found via a natural river in its ways from the starting place (source) to the destination which results in eventually finding a very good path to their destination. A comprehensive computational and statistical analysis is conducted to analyse the algorithms’ performances. Experimental results reveal that the proposed hybrid IWD algorithm is a trustable and proficient one in finding very good solutions, since it is already proved that the IWD algorithm has the property of the convergence in value.

Keywords: earliness and tardiness; intelligent water drops (IWD) algorithm; unrelated parallel machines; makespan; sequence-dependent set-up time (SDST)

1. Introduction

Parallel machine scheduling is important from both the theoretical and practical points of view. Since parallel machine environments include a subset of more complicated machine environments, it is important from a theoretical viewpoint. Besides, many algorithms in parallel machine environment could be reduced to solve single-machine problems. Also, since in a real world, most of facilities/factories have multiple machines/processors, this environment is important from a practical perspective. In the literature, there are three types of parallel machines scheduling, however most of researches are limited to situations in which the processing times are the same across all machines. This type is called identical parallel machines, $P_m$. In the second type, machines have different speeds but each machine works at a consistent rate, $Q_m$ (Pinedo 2008; Potts 1985). Finally, when machines are capable of working at different rates and when different jobs could be processed on a given machine at different rates, the environment is said have unrelated parallel machines, $R_m$ (Pinedo 2008). Developed algorithms to schedule unrelated parallel machines are capable of generating good solutions when applied to all kinds of parallel machine problems and that’s why they are important. Pfund, Fowler, and Gupta (2004) pointed out that unrelated parallel machine problems remain relatively unstudied and presented a survey of algorithms for single-and multi-objective unrelated parallel machine deterministic scheduling problems. There is much research work considering parallel machines, but few unrelated parallel machines or sequence-dependent set-up times (SDST). In general, such a problem is composed of job allocation and job sequencing onto the machines, simultaneously, with similar but not necessarily identical capabilities.

Considering both earliness and tardiness simultaneously in scheduling environments has been motivated by the just-in-time (JIT) philosophy in recent decades. JIT production has proved to be an important requirement of world class manufacturing due to the significant increase in the competitive productive world in the last decades. Identifying and eliminating waste components during the production process such as waiting time, transportation, inventory, movement and defective products are included in JIT philosophy (Wang 2006). The JIT approach requires only the necessary units to be provided with the necessary quantities, at the necessary times. Producing one extra unit is as bad as being one unit...
short (Omar and Teo 2006). Since, neither earliness nor tardiness is desirable, because one can represent manufacturer concerns and the other one may represent the customer concerns, this research tries to minimise earliness and tardiness as well as makespan, simultaneously. On the other hand, makespan is one of the most widely studied objectives in the literatures and reducing it is of great interest (Eswaramurthty 2008). Also, makespan is a measure of utilisation and relates directly to the system’s efficiency, i.e. the efficiency increases as the makespan decreases. In line with rising current trends towards JIT policy, the conventional performance measures are no longer applicable. In its place, the emphasis has shifted towards E/T scheduling taking earliness in addition to tardiness into account (Baker and Scudder 1990). Baker and Scudder (1990) presented the first survey on E/T scheduling problems. Also it should be pointed out the E/T problem has proven to be NP-hard (Hall and Posner 1991; Sun and Wang 2003).

Since parallel machines have been proven to be NP-hard (Pinedo 2008) in many situations, a ‘good’ solution obtained by a heuristic (meta-heuristic) algorithm in reasonably short computational time is often desirable. A meta-heuristic algorithm could be classified as a constructive approach or a local search method. The solutions could be constructed from scratch by piecemeal appending solutions’ components to the initially empty solutions in a constructive algorithm while a local search initiates with a complete solution and makes an attempt to enhance it over time. In this paper, a new swarm-based nature-inspired optimisation method, called intelligent water drops (IWD) is developed on the unrelated parallel machines. The objective is to minimise earliness and tardiness as well as makespan. The IWD is one of the recently proposed algorithms in the field of the swarm intelligence (Shah-Hosseini 2007) which is based on the dynamics of river systems, actions and reactions that happen among the water drops in rivers. The IWD algorithm is naturally appropriate for combinatorial optimisation problems (Shah-Hosseini 2012) and it has been shown to have the property of convergence in value (Shah-Hosseini 2008). The IWD algorithm has been tested using several standard optimisation benchmark problems. It can find good solutions for Travelling Salesman Problem (TSP) (Shah-Hosseini 2007, 2009) and it can also solve robot path planning (Duan, Liu, and Lei 2008; Duan, Liu, and Wu 2009), the n-queen puzzle (Shah-Hosseini 2005) and the multiple knapsack problem (MKP) (Shah-Hosseini 2009) with optimal or near-optimal solutions. To the best of the authors’ knowledge, this research is the first to apply the IWD algorithm to parallel machine scheduling problem.

The rest of paper is organised as follows: in the next section, literature review is presented. Problem description is expressed in Section 3. Section 4 explains the IWDs algorithm. In Section 5, we explain how this algorithm is applied on the parallel machines. The proposed hybrid IWDs algorithm with variable neighbourhood search (VNS) technique is presented in the subsequent section. Computational results and implementation details are demonstrated in Section 6. Lastly Section 7 discusses conclusion remarks and future works.

2. Literature review

As pointed out earlier, the unrelated parallel machines have been less studied with SDSTs and only some papers could be found in the literature. Of them, one could mention to Logendran, McDonell, and Smucker (2007) which proposed a methodology for minimising the weighted tardiness of jobs in unrelated parallel machining scheduling with sequence-dependent set-ups. Weng, Lu, and Ren (2001) addressed the unrelated parallel machine with job SDSTs to minimise a weighted mean completion time. Kim et al. (2002) proposed a simulated annealing (SA) to minimise the total tardiness. Minimising total weighted tardiness and the maximum lateness of jobs are studied in Kim, Na, and Chen (2003) and Kim and Shin (2003), respectively. The same problem with the objective to minimise makespan, maximum tardiness and total tardiness is studied in Chen (2005, 2006) and Chen and Wu (2006), respectively, where resource constraints are also considered. A heuristic for the unrelated parallel machines is proposed by Rabadi, Moraga, and Al-Salem (2006) with the objective to minimise makespan. Also, Rocha et al. (2008) studied unrelated parallel machines considering sequence and machine-dependent set-up times, due dates and weighted jobs and developed a branch-and-bound algorithm. They also provided a solution by the GRASP method which is used as an upper bound. Vallada and Ruiz (2011) presented a genetic algorithm (GA) for the unrelated parallel machine scheduling problem considering machine and job SDSTs. Lin, Pfund, and Fowler (2011) performed a study in which the performance of various heuristics is compared with one meta-heuristic for unrelated parallel machine scheduling problems. Their objective was to minimise makespan, total weighted completion time and total weighted tardiness.

Of papers which studied earliness and tardiness simultaneously on parallel machines, one could mention to Ventura and Kim (2003) which considered the problem of scheduling jobs on parallel machines with the objective of minimising total absolute deviation of job completion times about the corresponding due dates. Polyakovskiy and M’Hallah (2014) studied the weighted earliness, tardiness parallel machine problem where jobs have different processing times and due dates are distinct. They proposed a mixed integer programming (MIP) and solved it using a deterministic heuristic based on multi-agent systems. Drobouchevitch and Sidney (2012) solved a scheduling problem of identical non-preemptive
jobs with a common due date on uniform parallel machines. Their objective was to determine an optimal value of the due date as well as an optimal jobs allocation onto machines in order to minimise the total cost function including earliness, tardiness plus due date values. Also, Janiak et al. (2013) studied identical parallel machines with the goal of finding a job schedule plus location and size of the due window such that total incurred cost is minimised, i.e. earliness, tardiness, due window location and size associated costs. M’Hallah and Al-Khamis (2012) studied minimising total weighted earliness and tardiness on parallel machines. They used an exact MIP for small-sized cases; whereas, larger instances are solved using hybrid algorithms. Widely used performance measures in due date-related scheduling problems include maximum tardiness, total or mean tardiness, total weighted tardiness/earliness and the number of tardy jobs (Koulamas 1994; Sen, Sulek, and Dileepan 2002). Yi and Wang (2003) addressed a model for scheduling grouped jobs on identical parallel machines with the objective of minimising total tardiness and earliness penalties. Cheng and Sin (1990) studied a comprehensive review on parallel machine scheduling problems with conventional performance measures based on due date, completion time and flow time. Toksari and Guner (2009) considered a parallel machine earliness/tardiness scheduling problem with different penalties under the effects of position-based learning and linear and non-linear deteriorations. As indicated by Morton and Pentico (1993) and Liaw et al. (2003) due date-related problems for multi-machine environments are usually computationally complex and hence most existing results are typical for problems with small sizes or simple settings. For example, Pinedo and Singer (1999) and Yang, He, and Cho (1994) used the shifting bottleneck method, which was originally designed to minimise makespan, to decompose the multi-machine problems into a series of single-machine problems. Su (2009) addressed the identical parallel machine scheduling problem in which the total earliness and tardiness about a common due date are minimised subject to minimum total flow time. Mason, Jin, and Jampani (2009) developed parallel machine moving block heuristic (Pm-MBH) with polynomial-time computational complexity to address earliness–tardiness problem on identical parallel machines. They presented a MIP to analyse the solution quality of the Pm-MBH solutions. Radhakrishnan and Ventura (2000) studied parallel machine earliness–tardiness non-common due date sequence-dependent set-up time scheduling problem for jobs with varying processing times. Their objective was to minimise the sum of the absolute deviations of job completion times from their corresponding due dates.

Bank and Werner (2001) considered unrelated parallel machine regarding release date as well as common due date. Their objective was to distribute the jobs to the machines and to schedule the jobs assigned to each machine such that the weighted sum of linear earliness and tardiness penalties is minimal.

Of researches that have applied the IWD algorithm on optimisation problems, one could name Shah-Hosseini (2007, 2009), which applied the IWD on the TSP. Also, Duan, Liu, and Lei (2008) and Duan, Liu, and Wu (2009) proposed an improved IWD optimisation algorithm for solving the air robot path planning problems in various environments. Shah-Hosseini (2009) tested the IWD algorithm to find solutions of n-queen puzzle. Also, MKP was tackled in Shah-Hosseini (2009) and tried to find optimal or near-optimal solutions. Niu, Ong, and Nee (2012) customised the IWD for solving job-shop scheduling problems. They proposed five schemes to improve the original IWD algorithm. Niu, Ong, and Nee (2013) solved the multi-objective job-shop scheduling problem with the goal of finding the best compromising solutions (Pareto non-dominance set) considering multiple criteria, i.e. makespan, tardiness and mean flow time of the schedules.

In addition to the aforementioned areas, the IWD algorithm has also been applied to different optimisation problems in different fields of research. It is shown that the IWD gives better or at least comparable consequence in comparison with the other well-known optimisation methods, such as PSO, ACO, etc. (Niu, Ong, and Nee 2013). Of these works, one could mention to Abbasy and Hosseini (2008), which solved the economic dispatch and emission dispatch problems in power systems. In the field of transportation, distribution and logistics, the vehicle routing problem (VRP) was studied by Kamkar, Akbarzadeh-T, and Yaghoobi (2010). Also, textual features selection for developing precision irrigation system was addressed in Hendrawan and Murase (2011). Dariane and Sarani (2013) applied the IWD on the reservoir operation problem which is a challenging problem in water resources systems. Their computational results showed that despite the IWD algorithm finds relatively higher quality solutions, it is also able to conquer the computational time consumption deficiencies inherited in the ACO algorithm. The latter case is very significant in large-scale problems with a large amount of decision variables where the computational time plays a limiting factor role for the model.

3. Problem description

In the considered parallel machine scheduling problem with sequence-dependent set-up times, all parameters are supposed to be deterministic. A set of $N$ jobs denoted by $1, 2, \ldots, n$ has to be processed on a set of $M$ unrelated parallel machines denoted by $M_1, M_2, \ldots, M_m$ in this study. A MIP mathematical model with the goal of minimising total earliness and tardiness in addition to makespan is proposed for the considered problem. The set-ups are assumed to be
simultaneously machine- and job-dependent, consequently a set-up time \( S_{kim} \) is incurred, when a given job \( k \) is processed immediately after job \( i \) on machine \( j \). All assumptions as well as parameters, decision variables and mathematical model are as follows.

### 3.1 Assumptions

- Only one operation could be processed on each machine at a time.
- After starting the process by machine, no idle time could be inserted into the schedule.
- All jobs and machines are available in time zero.
- All machines are unrelated and each job could be processed by any free machine.
- Number of jobs and machines are constant.
- All processing times and due dates are deterministic and predefined.
- The process time of each job on each machine differs for each other.
- All machines are not capable of processing all jobs and operations.
- The set-up time for each job on each machine is sequence-dependent.
- No preemption of operations of each job is allowed.
- Transportation time between machines is negligible.
- Each job has a distinct due date and must be processed only one time.
- Machines are available throughout the scheduling period (i.e. no breakdown).

### 3.2 Notations

#### 3.2.1 Subscripts

- \( N \) Number of jobs
- \( M \) Number of machines
- \( i, k \) Index for job \((i, k = 1, 2, \ldots, N)\)
- \( j \) Index for priorities \((j = 1, 2, \ldots, J)\)
- \( m \) Index for machine \((m = 1, 2, \ldots, M)\)

#### 3.2.2 Input parameters

- \( p_{im} \) Processing time of job \( i \) on machine \( m \)
- \( \alpha_i \) The earliness unit penalty of job \( i \)
- \( \beta_i \) The tardiness unit penalty of job \( i \)
- \( d_i \) Due date of job \( i \)
- \( \lambda \) Factory costs per time unit (including machines, labour and variable production costs and the costs dependent to the work time)
- \( S_{kim} \) Set-up time for assigning job \( i \) after job \( k \) on machine \( m \)

#### 3.2.3 Decision variables

- \( C_i \) Completion time of job \( i \)
- \( C_{\text{max}} \) Total completion time or makespan
- \( E_i \) Earliness of job \( i \); \( E_i = \max\{0, d_i - C_i\} \)
- \( T_i \) Tardiness of job \( i \); \( T_i = \max\{0, C_i - d_i\} \)
- \( y_{ijm} \) 1 if job \( i \) on machine \( m \) in priority \( j \); otherwise, it is zero

### 3.3 The mathematical model

\[
\min Z = \min \left( \lambda C_{\text{max}} + \sum_{i=1}^{N} (\alpha_i E_i + \beta_i T_i) \right) \tag{1}
\]
Equation (1) shows the objective function which aims to minimise total weighted earliness and tardiness as well as makespan.

Subject to

\[
\sum_{k=1}^{N} y_{0km} = 1 \quad \forall m;
\]

Equation (2) ensures that dummy job 0 must be processed earlier than any other job on each machine and after that the other jobs could be processed.

\[
\sum_{k=1}^{N} y_{ikm} \leq b m \quad \forall m, i \in \{1, \ldots, N\}, i \neq k;
\]

Inequality (3) makes sure that only one job after another (utmost one job) could be processed on each machine, where this machine is capable of processing this new job.

\[
\sum_{m=1}^{M} \sum_{i=0}^{N} y_{ikm} \leq 1 \quad \forall k \in \{1, \ldots, N\}, i \neq k;
\]

Constraint (4) ensures there was utmost only one job before processing another job on the same machine. In fact, Constraints (3) and (4) jointly guarantee that there is only one job before and one job after the current given job on the same machine.

\[
\sum_{m=1}^{M} \sum_{i=0}^{N} \sum_{k=1}^{N} y_{ikm} = N \quad i \neq k;
\]

Equation (5) ensures all jobs must be processed on machines and there will be no unassigned job among all jobs.

\[
\sum_{j=0}^{N} y_{ikm} \geq \sum_{j=1}^{N} y_{ikm} \quad \forall m, k \in \{1, \ldots, N\}, i \neq k, j \neq k;
\]

Inequality (6) guarantees that job \( i \) could be processed on only one machine.

\[
C_i - d_i = T_i - E_i \quad \forall i;
\]

The earliness and tardiness of job \( i \) is defined using Constraint (7).

\[
y_{i1m} \times p_{im} \leq C_i \quad \forall i, m;
\]

\[
\left( \sum_{m=1}^{M} \sum_{j=2}^{J} \sum_{k \neq i}^{N} y_{kj1m} \cdot y_{ijm} (C_k + S_{km}) \right) + \sum_{m=1}^{M} \sum_{j=1}^{J} p_{ijm} \cdot y_{ijm} = C_i \quad \forall i;
\]

Constraints (8) and (9) together guarantee that only after starting the process by machine, no idle time could be inserted into the schedule and also no job preemption is allowed. 

\[
C_{\text{max}} \geq C_i \quad \forall i;
\]

\[
y_{ijm} \in \{0, 1\} \quad \forall i, j, m;
\]

\[
T_i, E_i, C_i \geq 0 \quad \forall i;
\]

The maximum completion time is defined via Inequality (10). Lastly, Set (11) defines the binary variables and Set (12) identifies non-negativity constraints.

Given the above system description, the objective is to simultaneously determine the job-machine assignments and the jobs sequencing on each machine, so as the total weighted earliness and tardiness as well as makespan can be minimised. The considered properties of this problem, to a large extent, make it more realistic. The linearisation of the proposed mathematical model is affixed in Appendix 1.
4. IWDs algorithm

4.1 An overview of IWDs

IWDs algorithm is inspired from natural rivers and is a population-based nature-inspired optimisation algorithm. This algorithm could be employed to solve a large variety of optimisation problems (Shah-Hosseini 2009). The IWD finds very good path to the rivers’ destination despite many different kinds of obstacles on their ways. A natural river often finds good paths among plenty of possible paths in its ways from the starting place (source) to the destination. A very good solution (path) could be found regarding the conditions of its surroundings to attain to its final destination which is often a lake, a sea or an ocean. The good solutions follow from actions and reactions happening among the water drops and the water drops with the riverbeds. In the way towards the final destination, several artificial water drops influence the environment around as they move through the river bed. This flow towards the destination is caused by the gravitational force of the earth. Suppose that there are no obstacles/barriers in the water drops path. So, they would obviously follow a straight path towards the destination, which is absolutely the shortest path from the starting place to the destination. However, since in real-world conditions there are a lot of turns, twists and other types of obstacles/barriers in the river path, the real path may be consequently different from the ideal one. It’s noticeable that this constructed path sounds to be optimum considering the distance from the destination and the environment constraints (2008; Duan, Liu, and Wu 2009).

The IWD has two main features: (1) the amount of the soil it carries now, soil (IWD) and (2) the velocity which it is moving now, velocity (IWD). The velocity makes the water drops possible to transfer soil from one place to another. Faster water drops could obviously gather and transfer more soil from the river beds. Those parts of bed river get deeper by being removed from soil, more water could be attracted since they could hold more amounts of water. The removed soils, which are carried in the water drops, are unloaded in slower river beds. The IWDs’ soil collection and movement are affected by the soil amount in each path. In fact, each IWD holds soil in itself and removes soil from its path during movement in the environment. The less-carried amount of soil, the faster the water drops speed and the IWDs could consequently collect more soil from that path, while a path with more soil is the opposite. As mentioned above, the removed soil amount from the path is determined by the velocity of an IWD flowing over a path. On the contrary, the IWD velocity is also changed by the path such that a path with little soil amount increases the IWD velocity more than a path with a notably large amount of soil.

The IWD environment depends on the problem at hand. Generally, there are often many paths from a given source to a desired destination in an environment where the position of the destination may be known or unknown. If the position of the destination is given, the objective is obviously to find the best (shortest) path from the starting place to the destination. In those cases with unknown destination, the objective is to find the optimum destination which may be interpreted as cost or any appropriate measure for the on-hand problem. In the IWD algorithm, the movement of IWDs from source to destination is carried out in discrete finite-length time steps. By moving IWDs from one location to the next, the velocity increment is non-linearly proportional to the inverse of the soil between the two locations. In such a condition, the IWDs soil is increased since some soil is removed from the path. The soil amount increment is inversely proportional to the time needed for the IWDs to pass from its current location to the next location. Also, the time duration to travel from a given location to the next location is calculated according to the physics law for linear motion, i.e. the time taken is proportional to the distance between the two locations and inversely proportional to the velocity of the IWD.

In the IWD algorithm, the soil amount on the path is interpreted as hardness. A branch of the path is less desirable, if it contains higher soil amounts than other branches. This mechanism for branch selection on the path is implemented by a probabilistic function of inverse of soil. Also, a water drop prefers an easier path to a harder one when it has to choose between several branches which exist in the path from the starting place to the destination. In the IWD algorithm, a parameterised probabilistic model is employed to construct solutions. Also, the parameters values are updated so as to enhance the probability of constructing high quality solutions.

It is already proved that IWD has the convergence property in value (Shah-Hosseini 2008), which means that the IWD algorithm is capable of finding the optimal solution if the number of iterations is sufficiently large. In this context, if it is proved that there is positive chance for any feasible solution to be found by an IWD in an iteration of the algorithm, it will be guaranteed that the optimal solution is found, since once an IWD finds an optimal solution, that solution becomes the iteration-best solution in the algorithm and consequently the total best solution is updated to the newly found optimal one. This fact is well proved in Shah-Hosseini (2008) in terms of proposition 5.2. Accordingly, the IWD algorithm is appropriate for a large variety of applicable problems such as VRP, TSP and Robot path planning, and scheduling problems such as parallel machine ones which are investigated in this research.
There are three reasons which provide the importance and necessity of the IWD algorithm:

1. It provides good quality solutions using average values.
2. Higher convergence speed compared to other methods.
3. Flexibility in the dynamic environment and incorporating pop-up threats easily.

The IWD algorithm could be compared to the ant-based optimisation algorithms (Bonabeau, Dorigo, and Theraulz 1999). The ants in an ACO algorithm put pheromones on the paths they move on, while the IWDs change soil on the paths they flow over. Although, the made changes in IWD algorithm are not steady and are dependent on two factors, i.e. (1) velocity and (2) soil of the IWD visiting the paths which are in contrast to the ants in ACO. Furthermore, the IWDs may get different velocities during an iteration of the IWD algorithm while the ant’s velocities are irrelevant to the algorithm in ant-based algorithms (Shah-Hosseini 2009). Besides, for a few particular ACO algorithms and careful ACO parameter setting, ‘finding the optimal solution’ property has been shown to exist and this type of convergence is called convergence-in-value (Dorigo and Stutzle 2004).

4.2 IWDs algorithm: an optimisation method

This study proposes an IWD algorithm, suggested by Shah-Hosseini (2007), for parallel machine scheduling with SDSTs jobs and distinct due dates. An overview of IWD is described in the previous subsection. As pointed out earlier, there are two important features in the IWD algorithm:

1. The amount of the soil it carries now, soil (IWD).
2. The velocity which it is moving now, velocity (IWD).

As the IWDs flow in an environment, both velocity and soil may change. A given environment corresponds to a problem which is desired to be solved. The IWDs River seeks very good path for the known problem. There are several paths from a specified source to a desired destination in a given environment.

The considered environment in which IWDs are moving is assumed to be discrete. This environment consists of $N$ nodes and $E$ edge, denoted as graph $(N,E)$ in which each IWD should move from one node to another. Every two nodes are connected by an arc which holds a soil amount. The soil of each arc may be changed according to the IWDs activities flowing in the environment. In fact, by travelling on the graph nodes along the edges of the graph, each IWD begins to construct its solution slowly until the IWD completes its solution. By completing all IWDs’ solutions, an iteration of the algorithm is performed.

By this way, the set of best (elite) solutions in each iteration, $S_e$, is found by running each iteration and is employed to update the best solution obtained so far, GBS. Also, the soil amount on the edges of $S_e$ is decreased based on the solutions’ goodness. The $S_e$ as well as a generated random set of IWDs ($S_p$) entirely called $S_{po}$ of size $N_{po}$ are selected as the initial solution of VNS method, as a well-known local search technique (post-optimisation). Thereafter, the soil amount of $S_e$ and GBS are again updated. The algorithm initiates another iteration with new IWDs, however, with the same soils on the graph paths and subsequently the entire process is repeated. Two criteria could be defined as stop criteria in this algorithm, i.e. (1) the maximum number of iterations, (itermax) and (2) the GBS reaches the expected quality.

There are two types of parameters in the IWD algorithm. The first type, which is called ‘static parameters’, remains constant during the algorithm lifespan and the second type is dynamic which is reinitialised after each iteration. The used method to choose the values of the static parameters of the IWD algorithm is the same in (Shah-Hosseini 2008) until specified otherwise.

5. IWDs algorithm on unrelated parallel machines

An IWDs algorithm is proposed in this study to apply on parallel machine scheduling problem in which the machines are unrelated. In such an area, the goal is to assign and sequence $n$ jobs over $m$ machines to minimise the total earliness and tardiness of jobs as well as maximum completion time or makespan. To solve such a problem using IWD algorithm, two decisions should be made: (1) determining the assignment of jobs to machines and (2) determining the jobs order to obtain a good solution which will minimise the considered objective.

The steps of IWD algorithm in each iteration are as follows:

1. **Initialisation**: all static and dynamic parameters such as each edge’s soil and each IWD’s velocity are firstly initialised. As a matter of fact, all the IWDs have the same initial velocity and all the edges are set with the same amount of initial soil in the original IWD algorithm.
Selection of next node: for each IWD, using roulette wheel mechanism select randomly the next node to build a solution (path) as it will be described in Section 5.1 according to the probability \( \text{Pro}_{IWD}^{j} \):

\[
\text{Pro}_{IWD}^{j}(j) = \frac{f(\text{soil}_{i,j})}{\sum_{v \in \text{Edges}} f(\text{soil}_{v,n})}
\]

(13)

\[
f(\text{soil}_{i,j}) = \frac{1}{e + g(\text{soil}_{i,j})} \quad e \text{ is a small positive value}
\]

(14)

\[
g(\text{soil}_{i,j}) = \begin{cases} 
\text{soil}_{i,j} & \text{if } \min_{v \in \text{Edges}}(\text{soil}_{v,n}) \geq 0 \\
\text{soil}_{i,j} - \min_{v \in \text{Edges}}(\text{soil}_{v,n}) & \text{else}
\end{cases}
\]

(15)

Update Velocity: Update each IWD’s velocity moving from node \( i \) to node \( j \) on the disjunctive graph, as follows:

\[
\text{Velocity}_{IWD}^{j} = \text{Velocity}_{IWD}^{j-1} + \frac{a_e}{b_v + c_v \cdot \text{soil}_{i,j}^{j-1}} \quad \forall i \text{ traversed before } j
\]

(16)

Compute Delta Soil: for each IWD, compute the soil amount which it loads from the edge \((i, j)\) as follows:

\[
\text{time}_{IWD}^{j} = \frac{\text{Dist}_{i,j}}{\text{Velocity}_{IWD}^{j}}
\]

(17)

\[
\Delta \text{soil}_{i,j}^{j} = \frac{a_s}{b_s + c_s \cdot (\text{time}_{IWD}^{j})^2}
\]

(18)

where \( \text{Dist}_{i,j} \) is the undesirability of an IWD to go from the current node \( i \) to the next node \( j \) in the generated tour as it will be figured out in Section 5.1.

Update Edge Soil and IWD Soil: for each IWD, update the edge soil traversed by that IWD and the soil included in the IWD:

\[
\text{soil}_{i,j}^{k+1} = (1 - \rho) \cdot \text{soil}_{i,j}^{k} - \rho \cdot \Delta \text{soil}_{i,j}^{k}
\]

(19)

\[
\text{soil}_{IWD}^{k+1} = \text{soil}_{IWD}^{k} + \Delta \text{soil}_{i,j}^{k}
\]

(20)

Set up Elite Set: Constitute a set of elites solutions of size \( N_e \).

Global Soil Propagation: update the edge soil included in the current elite IWDs’ solutions \( N_e \):

\[
\text{soil}_{i,j} = (1 - \rho_{IWD}) \cdot \text{soil}_{i,j} - \frac{\rho_{IWD}}{d - 1} \cdot \text{soil}_{i,j}^{IWD}; \quad \forall (i,j) \in S^{\text{elite}}, \quad k \in [1, \ldots, N_e]
\]

(21)

where \( d \) addresses the node numbers in predefined network.

Set up Post Optimisation Set: a solution set \( S_{po} \) is constituted so as to find better local optimums via a strong neighbourhood search, i.e. VNS.

\[
S_{po} = S_{e} + S_{r}; \quad e \in \{1, \ldots, N_e\}; \quad r \in \{1, \ldots, N_r\}; \quad po \in \{1, \ldots, N_{po}\}
\]

(22)
9. **Embedded Local Search**: this function is used as a post optimisation technique to generate higher quality solutions through employing VNS algorithm. In better words, the VNS uses the solutions of $S_{po}$ as a good initial solution and tries to enhance its quality. This improving further occurs in this study since an appropriate neighbourhood structure is defined. Supplementary clarifications are explicated in Section 5.2.

10. **Update (GBS)**: is employed to update the global best solution, $GBS$, via the best iteration solution (the best solution in $S_{c}$, called $T_{Se}$) as follows:

\[
GBS = \begin{cases} 
    GBS; & \text{if } q(GBS) > q(T_{Se}) \\
    T_{Se}; & \text{else}
\end{cases}
\]

where $q(GBS)$ is a quality function and is defined as the fitness of the given schedule, i.e. the smaller earliness and tardiness as well as the makespan, the better fitness.

5.1 **Construction of solution**

To illustrate the construction of the feasible solution on the parallel machines, the jobs are represented as square in this study (which could also be defined as node clusters). Each square has $m$ nodes which correspond to the machines on which each job could be processed. Despite the nodes on each square are not connected, however, every node in a square is connected to all other nodes in other squares. A dummy node, which could be regarded as the start/end point of a water drop’s tour, is defined and connected to every other node on the graph. The defined graph has $n$ squares, $m \times n$ nodes as well as one dummy node. The total number of edges are $(nm)^2$. To construct a solution, the artificial water drop travels to all squares (i.e. visits only one node [machine] on that square) and after that returns back to the dummy node. The latter action completes the tour on the defined graph. In this solution construction, all jobs must be assigned on machines where each job could only be processed on one machine. This tour spans $(n+1)$ edges. By completing a tour via water drop, the order of visiting each square yields the order of jobs assignment to the machines. Also, the node visited at each square determines the job assignment to a machine. Figure 1(b) illustrates such solution construction. In this schematic figure, there are six jobs and four machines.

The used solution construction scheme of parallel machines scheduling problem in IWD (Figure 1(a)) is generated as follows: The water drop’s tour (depicted by dotted) is (0), (1,3), (2,2), (6,3), (5,4), (4,1), (3,1) and (0) where the pair $(i, k)$ denotes (job, machine). The solution corresponding to this path is as follows: jobs 4 and 3 are first processed on machine 1, respectively. There’s only job 2 assigned on machine 2. On machine 3, jobs 1 and 6 are processed, respectively. Lastly, job 5 is processed on machine 4. This representation scheme could be demonstrated in another form, i.e. the water drop’s route could be changed subject to not changing the sequence of those jobs which are processed on the same machine. In better words, suppose that the water drop’s tour is as follows: (0), (1,3), (5,4), (6,3), (2,2), (4,1), (3,1) and (0). As could be seen, the sequence of jobs 3 and 4 on machine 1 is not changed; however, job 5 is processed on machine 4 earlier than previous sequence. This sequence interchangeability, to a large extent, makes higher quality-obtained solutions, since by constituting different seeds, the VNS is capable to explore a vast area and this is a considerable advantageous.

![Figure 1](image-url)  
Figure 1. Solution representation of the considered problem on IWD.
5.2 The proposed VNS embedded in IWD

The VNS proposed by Mladenović and Hansen (1997) and is a relatively new local-search-based meta-heuristic where a large number of successful applications have been reported (Hansen and Mladenović 2001). The main idea is to improve a simple local search process to make it possible escaping from local optima which is carried out by regenerating the local search from a randomly chosen neighbour of the incumbent solution. Since this method employs two or more neighbourhoods in its structure, instead of one, it differs from the most local search heuristics; especially as it is based on the rule of systematic change of neighbourhood during the search. Moreover, the best number of neighbourhoods is often three (Rocha et al. 2007) so as to keep away from spending a huge computational time.

The three used neighbourhood structures in this study are as follows:

1. **Swapping jobs on one machine**: one machine is firstly selected and all possible job swaps on it are taken into account \((N_1(x))\).
2. **Swapping jobs between two different machines**: two machines are selected and all possible job swaps from these different machines are regarded \((N_2(x))\).
3. **A given job moves from one machine to another**: one machine is selected and all possible job movements from this machine to any other are considered \((N_3(x))\).

It should be pointed out that the above-mentioned neighbourhoods are determined according to its correspondent structures, and the solution it is being applied to. Also, the size of neighbourhood \(N_1(x)\) is \(O(m^2 n^2)\), neighbourhood \(N_2(x)\) is \(O(m^2 n^2)\) and neighbourhood \(N_3(x)\) is \(O(n^2)\). Figure 2 shows the basic VNS structure.

The algorithm endeavours constantly to employ the fastest local search available first. If no improvement is made after an iteration, another neighbourhood is then applied \((k\) is incremented). Also whenever a new solution is found, the first and fastest local search is employed \((k = 1)\). For further reading about neighbourhood search methods, one could be referred to Ahuja et al. (2002).

5.2.1 Random solution

Once a neighbourhood is selected, a random procedure is called. A random solution is chosen via this procedure from the selected neighbourhood structure. Three procedures are then constructed as follows:

1. \(N_1(x)\):
   (a) Select a given machine \(i\) randomly
   (b) Select two jobs \(j_1\) and \(j_2\) randomly
   (c) Swap jobs \(j_1\) and \(j_2\)

2. \(N_2(x)\):
   (a) Select two machines \(i_1\) and \(i_2\) randomly
   (b) Select a job \(j_1\) from \(i_1\) and a job \(j_2\) from \(i_2\) randomly
   (c) Swap jobs \(j_1\) and \(j_2\)

---

**Algorithm 1: Basic VNS Structure**

```plaintext
begin
initialization Select the set of neighborhood structures \(N_k\), \(k=1,...,k_{max}\);
Find an initial solution \(x^*\); Choose a stopping criteria (running time, number of iteration, etc.);
for \(i=1\) to \(max-itr-vns\) (max number of iteration in VNS)
   \(x^* : x^*\);
   Shake procedure: find a random solution \(x \in N_0(x)\);
   Perform a local search on \(N_0(x)\) to find a solution \(x^*\);
   if \(f(x^*) \leq f(x^*)\) then
      \(x^* : x^*\);
   continue with \(i = 1\);
end if;
end for;
end;
```

Figure 2. The pseudo-code of VNS algorithm.
Algorithm 2: Local search (I)

for each \( i \) do
    for each \( j_1 \) in \( i \) do
        for each \( j_2 \) in \( i \) s.t. \( j_1 \neq j_2 \) do
            if solution considering \( j_1 \) and \( j_2 \) swapped < current solution then
                Swap \( j_1 \) and \( j_2 \);
            end if;
        end for;
    end for;
end for;

Figure 3. The pseudo-code of local search (I).

\( N_3(x) \):

(a) Select a given \( j_1 \) as well as a machines \( i_2 \) randomly, where \( j_1 \) does not belong to \( i_2 \)
(b) Select a valid position, called ‘pos’ in \( i_2 \), randomly
(c) Transfer job \( j_1 \) to \( i_2 \) at the position ‘pos’

5.2.2 The local searches in VNS

Among different versions of VNS structures, a specific local search is employed for each neighbourhood in this study which is stated as follows:

Local search (I) – job swaps at a given machine: Every possible swap on this given machine is investigated via this local search. This procedure has a time complexity of \( O(mn^2) \) which comes in Figure 3.

Figure 4 shows the performance of local search (I). In this figure, jobs J12 and J15 are swapped on machine 2.

Local search (II) – job swaps on different machines: All jobs swaps between jobs belonging to different machines are assessed. A wider range of solutions are explored in this local search than local search (I), where its time complexity is \( O(m^2n^2) \). This local search is shown in Figure 5.

In Figure 6, jobs J17 and J4 are swapped on two different machines which demonstrate the local search (II)’s performance.

Local search (III) – job insertion: This method seeks for new solutions transferring jobs from the machine with the highest ‘objective function, i.e. total earliness and tardiness as well as makespan’ to the machine with the lowest one, where the time complexity of this procedure is \( O(n^2) \). This local search is shown in Figure 7 as follows:

In Figure 8, job J14 is removed from machine 2 and inserted to the machine 1 between jobs J3 and J4.
5.3 Hybrid intelligent water drops algorithm

The interest in the design and implementation of hybrid meta-heuristics has increased considerably in recent years (Talbi 2002). The hybrid meta-heuristics’ structure is motivated by the need to attain a good trade-off between the capabilities of a heuristic to investigate the search space and the possibility to take advantage of the experience gathered throughout the search. The proposed Hybrid Intelligent Water Drops algorithm (HIWD) in this paper, combining characteristics from IWDs algorithm and VNS technique, should not be considered as an endeavour to change a ‘main’ meta-heuristic into a hybrid one by grafting some contributions from other meta-heuristics. The HIWD permits the possible...
A combination of some basic characteristics from a reference set of meta-heuristics and aims at analysing the usefulness of the resultant customisable algorithm for a difficult problem as well as test instances.

In this section, an integrated hybrid algorithm, named HIWD, is developed. In each iteration, some local optimum solutions are generated via IWD algorithm and afterwards the VNS is employed for further enhancement of obtained

**Algorithm 5: The proposed HIWD for Unrelated Parallel Machines Scheduling Problem**

```plaintext
begin
initialization
Set the initial parameters: \( a_i, b_i, c_i, a_j, b_j, c_j, \theta, \rho, \delta \); 
Set the initial soil on bed: \( \text{init}_s\text{oil}(i, j) \text{ for all pairs of customers } i \text{ and } j \); 
Set the initial velocity of each water drop: \( \text{init}_v\text{el}(j) \); 
while predefined stopping criteria is not met do
   for each IWD do 
      current node \( i = \text{rand node } (j, k) \); # where the pair \((j, k)\) denotes (job, machine) #
      for each step \( t \) do # refer to Section 5: Eqs (11)-(18) #
         Calculate all possible paths \( \text{Pref}_{\text{IWD}}(j) \);
         Select next node \( j \) through the procedure Roulette wheel;
         Update Velocity\text{IWD};
         Update \( \text{soil}_{i,j} \);
         Update \( \text{soil}_{i,k} \);
         current node \( i = \text{next node } j \);
      end for;
      Evaluate IWD;
   end for;
   Update the elite set and its best solution \((TS) \);  
   Update the global best solution \((GBS) \); # Eq (21) #
   for each IWD in elite set do
      Update global soil;  # Eq (19) #
   end for;
   optimization set\( S_{opt} \) := \( N_r \text{ from elite set } (S) + N_r \text{ randomly from whole IWDs } (S) \); 
   for each IWD in post optimization set do
      procedure post optimization by Variable Neighborhood Search  # Algorithm 1 #
   end for;
   update the elite set and its best solution \((TS) \);  
   update the global best solution \((GBS) \);
   for each IWD in elite set do
      Update global soil;
   end for;
end while;
end;
```

Figure 8. Local search (III).

Figure 9. The pseudo-code of the proposed HIWD.
solutions. As a matter of fact, the VNS technique plays a role of post-optimiser in this hybrid algorithm. The algorithmic procedures of the proposed HIWD algorithm are demonstrated in Figure 9.

6. Computational results

6.1 Implementation details

In this section, we present the experimental results that validate the solution method design and demonstrate how it behaves for a varied set of instances. All instances are implemented in MATLAB 7.11.0 and run on a PC with a 3.4 GHz Intel® Core™ i7-2600 processor and 4 GB RAM memory. Two sets of test problems, i.e. small- and medium-to-large-size ones are implemented on such a problem.

The parameters used for the proposed algorithms in this study, i.e. IWD and HIWD are presented separately in right-hand side columns of Table 1. Since it is expected that pure IWD behaves differently with respect to HIWD, it is tuned somewhat various. That is, since the HIWD is supposed to yield better results with respect to the pure IWD, accordingly a large amount of efforts should be made by pure IWD so as to overcome this lack and it needs to find a way to explore the search space in a more effective manner. To do so, one could increase both the number of iterations and number of IWDs for better exploring of solution space; however, in last iterations it should apply less probability of randomness due to final intensifications.

The initial values of the parameters are set based on theoretical studies and similar reported researches like Shah-Hosseini (2008). In better words, general parameters such as $a_v$, $b_v$ and $e$ are used according to Shah-Hosseini (2008) and Niu, Ong, and Nee (2012) and those parameters concerned to the characteristics of on-hand problem are tuned experimentally. In some cases, a larger value of some parameters will result in higher quality solutions, however by consuming a longer computation time; therefore, trade-off values are obtained based on experiments for these

### Table 1. Parameter values for the proposed IWD and HIWD.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Value of IWD</th>
<th>Value of HIWD</th>
</tr>
</thead>
<tbody>
<tr>
<td>$N_{-IWD}$</td>
<td>Number of IWDs</td>
<td>40–50</td>
<td>20–25</td>
</tr>
<tr>
<td>$N_{-Itr}$</td>
<td>Number of iterations in algorithm</td>
<td>100</td>
<td>40–60</td>
</tr>
<tr>
<td>$\text{max-itr-vns}$</td>
<td>Maximum number of VNS algorithm iterations</td>
<td>–</td>
<td>8–10</td>
</tr>
<tr>
<td>$a_v$</td>
<td>IWD velocity updating parameters used in Equation (16)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$b_v$</td>
<td>IWD velocity updating parameters used in Equation (16)</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>$c_v$</td>
<td>IWD velocity updating parameters used in Equation (16)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$a_s$</td>
<td>IWD soil updating parameters used in Equation (18)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$b_s$</td>
<td>IWD soil updating parameters used in Equation (18)</td>
<td>0.01</td>
<td>0.01</td>
</tr>
<tr>
<td>$c_s$</td>
<td>IWD soil updating parameters used in Equation (18)</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$\rho_l$</td>
<td>Local soil updating parameter used in Equation (19)</td>
<td>0.8–0.85</td>
<td>0.9</td>
</tr>
<tr>
<td>$\rho_{IWD}$</td>
<td>Global soil updating parameter used in Equation (21)</td>
<td>0.9–0.95</td>
<td>0.9</td>
</tr>
<tr>
<td>$\text{ini}_\text{soil}$</td>
<td>The initial soil on bed for whole paths</td>
<td>$U \sim [800–1200]$</td>
<td>$U \sim [800–1200]$</td>
</tr>
<tr>
<td>$\text{ini}_\text{velIWD}$</td>
<td>The initial velocity of each IWD</td>
<td>$U \sim [3, 4]$</td>
<td>$U \sim [3, 4]$</td>
</tr>
<tr>
<td>$\varepsilon$</td>
<td>A small positive value</td>
<td>0.001</td>
<td>0.001</td>
</tr>
<tr>
<td>$N_e$</td>
<td>Number of elite solutions</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>$N_r$</td>
<td>Number of randomly chosen solutions</td>
<td>5</td>
<td>8</td>
</tr>
</tbody>
</table>

### Table 2. Input parameters distribution.

<table>
<thead>
<tr>
<th>Input parameters</th>
<th>Distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing time ($p_i$)</td>
<td>$\sim DU[20, 70]$</td>
</tr>
<tr>
<td>Due dates ($d_i$)</td>
<td>$\sim U[d_{min}, d_{min} + pP]$</td>
</tr>
<tr>
<td>Earliness penalty ($a_{ij}$)</td>
<td>$\sim U(1.5, 3.5)$</td>
</tr>
<tr>
<td>Tardiness penalty ($b_{ij}$)</td>
<td>$\sim U(1.5, 3.5)$</td>
</tr>
<tr>
<td>Set-up times ($S_{\text{lim}}$)</td>
<td>$\sim DU[5, 40]$</td>
</tr>
<tr>
<td>Number of jobs ($n$)</td>
<td>$10–20–50$</td>
</tr>
<tr>
<td>Number of machines ($m$)</td>
<td>$2–3–5$</td>
</tr>
</tbody>
</table>
parameters. They are tuned by both trial-and-error and by designing experiments with different combinations of the parameters.

6.2 Test instances

The goal is to test the methods on generated instances on one hand and, on the other hand, to compare and analyse their performance on a variety of test problems, which should represent different planning systems and strategies. The input parameters’ information is shown in Table 2.

Where \( d_{\text{min}} = \max (0, P(v - \rho/2)) \) and \( P = 1/m \sum_{i=1}^{m} \sum_{j=1}^{n} p_{ij} \). The expression of \( P \) aims at satisfying the criteria of scale invariance and regularity described by Hall and Posner (2001) for generating experimental scheduling instances.

The two parameters \( v \) and \( \rho \) are the tardiness and range parameters, respectively. In this study, we consider \( v \in \{0.2, 0.5, 0.8\} \) and \( \rho \in \{0.2, 0.5, 0.8\} \). For each quadruple \((n, m, v, \rho)\) four instances are generated in which each case has run times in all methods so as to guarantee constancy of these techniques. Consequently, considering \( M = 2, 3 \) and \( 5, N = 10, 20 \) and \( 50, v = 0.2, 0.5 \) and \( 0.8, \) and finally \( \rho = 0.2, 0.5 \) and \( 0.8, \) for each scenario (combination of such quadruple) there are \( 3^4 \times 4 \times 5 = 1620 \) generated instances totally for each method.

6.3 Comparative results

In order to compare the obtained results via the proposed IWD, two well-known meta-heuristics, i.e. GA and SA are employed in medium-to-large-size problems. In fact, all generated instances are implemented via four algorithms, i.e.

Table 3. Computational results for small-size problems.

<table>
<thead>
<tr>
<th>( M )</th>
<th>( N )</th>
<th>( v )</th>
<th>( \rho )</th>
<th>Lingo CPU Time</th>
<th>SA CPU Time</th>
<th>( PRE_{\text{avg}} )</th>
<th>SA CPU Time</th>
<th>( PRE_{\text{avg}} )</th>
<th>GA CPU Time</th>
<th>( PRE_{\text{avg}} )</th>
<th>IWD CPU Time</th>
<th>( PRE_{\text{avg}} )</th>
<th>HIWD CPU Time</th>
<th>( PRE_{\text{avg}} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>10</td>
<td>0.2</td>
<td>0.2</td>
<td>10352.38</td>
<td>3.228</td>
<td>0.31</td>
<td>30.403</td>
<td>0.11</td>
<td>39.687</td>
<td>0.06</td>
<td>51.424</td>
<td>0.00</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>9644.92</td>
<td>5.785</td>
<td>0.32</td>
<td>27.863</td>
<td>0.08</td>
<td>38.099</td>
<td>0.07</td>
<td>47.858</td>
<td>0.03</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>11167.00</td>
<td>3.054</td>
<td>0.26</td>
<td>28.446</td>
<td>0.09</td>
<td>36.126</td>
<td>0.09</td>
<td>41.671</td>
<td>0.01</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>0.2</td>
<td>10693.86</td>
<td>5.952</td>
<td>0.29</td>
<td>36.890</td>
<td>0.09</td>
<td>48.130</td>
<td>0.10</td>
<td>61.936</td>
<td>0.02</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>12109.91</td>
<td>3.274</td>
<td>0.28</td>
<td>27.373</td>
<td>0.09</td>
<td>36.039</td>
<td>0.08</td>
<td>44.163</td>
<td>0.02</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>9829.25</td>
<td>2.660</td>
<td>0.24</td>
<td>31.087</td>
<td>0.07</td>
<td>38.483</td>
<td>0.11</td>
<td>48.362</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>10998.00</td>
<td>6.188</td>
<td>0.33</td>
<td>36.766</td>
<td>0.11</td>
<td>51.257</td>
<td>0.14</td>
<td>63.489</td>
<td>0.04</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean</td>
<td>11107.37</td>
<td>4.085</td>
<td>0.29</td>
<td>30.880</td>
<td>0.09</td>
<td>40.460</td>
<td>0.09</td>
<td>50.176</td>
<td>0.02</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td>10</td>
<td>0.2</td>
<td>0.2</td>
<td>10133.93</td>
<td>15.462</td>
<td>0.23</td>
<td>28.303</td>
<td>0.13</td>
<td>36.981</td>
<td>0.15</td>
<td>43.156</td>
<td>0.08</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>11107.86</td>
<td>8.858</td>
<td>0.32</td>
<td>30.128</td>
<td>0.19</td>
<td>39.557</td>
<td>0.13</td>
<td>45.329</td>
<td>0.02</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>11254.69</td>
<td>17.006</td>
<td>0.41</td>
<td>40.119</td>
<td>0.21</td>
<td>49.168</td>
<td>0.11</td>
<td>59.341</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>0.2</td>
<td>12049.88</td>
<td>15.925</td>
<td>0.29</td>
<td>49.613</td>
<td>0.14</td>
<td>68.850</td>
<td>0.15</td>
<td>84.546</td>
<td>0.10</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>12161.87</td>
<td>16.545</td>
<td>0.42</td>
<td>34.071</td>
<td>0.12</td>
<td>46.241</td>
<td>0.07</td>
<td>59.341</td>
<td>0.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>13429.53</td>
<td>27.129</td>
<td>0.38</td>
<td>51.730</td>
<td>0.19</td>
<td>63.101</td>
<td>0.11</td>
<td>63.489</td>
<td>0.04</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>0.2</td>
<td>12083.91</td>
<td>17.732</td>
<td>0.34</td>
<td>59.195</td>
<td>0.16</td>
<td>80.458</td>
<td>0.14</td>
<td>101.248</td>
<td>0.06</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean</td>
<td>12569.07</td>
<td>17.801</td>
<td>0.34</td>
<td>45.281</td>
<td>0.16</td>
<td>60.339</td>
<td>0.13</td>
<td>73.173</td>
<td>0.06</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>10</td>
<td>0.2</td>
<td>0.2</td>
<td>14294.52</td>
<td>41.589</td>
<td>0.47</td>
<td>57.566</td>
<td>0.22</td>
<td>80.002</td>
<td>0.26</td>
<td>94.101</td>
<td>0.14</td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>13767.69</td>
<td>39.815</td>
<td>0.38</td>
<td>56.834</td>
<td>0.18</td>
<td>71.445</td>
<td>0.19</td>
<td>80.698</td>
<td>0.11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>16196.93</td>
<td>46.755</td>
<td>0.35</td>
<td>58.536</td>
<td>0.25</td>
<td>75.837</td>
<td>0.15</td>
<td>89.455</td>
<td>0.08</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>16116.09</td>
<td>57.879</td>
<td>0.36</td>
<td>68.490</td>
<td>0.29</td>
<td>86.297</td>
<td>0.15</td>
<td>90.747</td>
<td>0.03</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>19014.24</td>
<td>54.100</td>
<td>0.33</td>
<td>78.930</td>
<td>0.29</td>
<td>89.041</td>
<td>0.17</td>
<td>86.714</td>
<td>0.09</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>15742.42</td>
<td>44.753</td>
<td>0.47</td>
<td>67.930</td>
<td>0.22</td>
<td>83.995</td>
<td>0.15</td>
<td>102.850</td>
<td>0.06</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>0.2</td>
<td>12717.20</td>
<td>42.496</td>
<td>0.41</td>
<td>71.706</td>
<td>0.23</td>
<td>93.274</td>
<td>0.23</td>
<td>94.073</td>
<td>0.12</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.5</td>
<td>16212.94</td>
<td>51.339</td>
<td>0.44</td>
<td>74.528</td>
<td>0.28</td>
<td>92.715</td>
<td>0.25</td>
<td>90.653</td>
<td>0.19</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>0.8</td>
<td>12933.14</td>
<td>69.873</td>
<td>0.42</td>
<td>67.652</td>
<td>0.24</td>
<td>84.150</td>
<td>0.21</td>
<td>93.986</td>
<td>0.13</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean</td>
<td>15221.69</td>
<td>49.844</td>
<td>0.40</td>
<td>66.908</td>
<td>0.24</td>
<td>78.084</td>
<td>0.20</td>
<td>91.475</td>
<td>0.11</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Mean of all</td>
<td>12966.04</td>
<td>23.910</td>
<td>0.34</td>
<td>47.690</td>
<td>0.17</td>
<td>61.628</td>
<td>0.14</td>
<td>71.608</td>
<td>0.06</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
SA, GA, IWD and HIWD. Generating initial population, crossover and mutation operators, selection mechanism and finally termination criterion in GA method is applied according to Vallada and Ruiz (2011). Also, the employed SA in this paper is exactly as the same as in Kim et al. (2002). In better words, generating initial population, neighbourhood solutions, initial temperature, cooling schedule, final temperature and stop criterion.

The mathematical model is firstly solved optimally for small-size cases (i.e. those problems with 10 jobs on 2, 3 and 5 machines) via Lingo 9.0. The used performance measure for small-sized instances is percentage relative error (PRE), since in this category the optimum solution could be found:

$$PRE = \frac{Algsol - O}{O}$$

where $Algsol$ is the objective value obtained by the selected heuristic and $O$ is the optimum value obtained by Lingo. The relative percentage deviation (RPD) is also employed in medium-to-large-size instances with the intention of comparing the efficiency of heuristic and meta-heuristic methods with respect to each other for all cases.

$$RPD = \frac{Algsol - Minsol}{Minsol}$$

where $Algsol$ is the objective value obtained by the selected heuristic and $Minsol$ is the best solution obtained for each instance. Lower values of $PRE$ and $RPD$ are perceptibly preferable. The computational results for small- and medium-to-large-size instances are shown in Tables 3 and 5. In these tables, the MCPU Time is mean of CPU Time for all cases and is calculated in seconds.

Some points should be mentioned based on the obtained results: as it could be seen, in small instances, the proposed HIWD algorithm, despite taking more computational time, provides solutions which deviate 6% over the optimum ones in average. In better words, the HIWD has the minimum gap with the optimum solution in all small cases. In some cases the HIWD yields the optimum solution which is a prominent point and shows the strength of this algorithm. It should be also pointed out that despite the IWD outperforms GA in ‘mean of all’, however, these algorithms yield

<table>
<thead>
<tr>
<th>Sum of squares</th>
<th>df</th>
<th>Mean square</th>
<th>F</th>
<th>Sig.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Between groups</td>
<td>1.131</td>
<td>3</td>
<td>0.377</td>
<td>103.677</td>
</tr>
<tr>
<td>Within groups</td>
<td>0.378</td>
<td>104</td>
<td>0.004</td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>1.509</td>
<td>107</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 4. ANOVA results for small-sized instances.

Figure 10. Means plot at the 95% confidence level HSD intervals in small-sized instances.
Table 5. Computational results for medium-to-large-size problems.

<table>
<thead>
<tr>
<th>M</th>
<th>N</th>
<th>v</th>
<th>ρ</th>
<th>SA CPU time</th>
<th>RPDavg</th>
<th>GA CPU time</th>
<th>RPDavg</th>
<th>IWD CPU time</th>
<th>RPDavg</th>
<th>HIWD CPU time</th>
<th>RPDavg</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>20</td>
<td>0.2</td>
<td>0.2</td>
<td>3.893</td>
<td>0.19</td>
<td>21.715</td>
<td>0.10</td>
<td>28.212</td>
<td>0.12</td>
<td>23.672</td>
<td>0.00</td>
</tr>
<tr>
<td>0.5</td>
<td>2</td>
<td>0.2</td>
<td>0.2</td>
<td>5.234</td>
<td>0.18</td>
<td>27.672</td>
<td>0.09</td>
<td>33.363</td>
<td>0.17</td>
<td>29.716</td>
<td>0.00</td>
</tr>
<tr>
<td>0.8</td>
<td>2</td>
<td>0.2</td>
<td>0.2</td>
<td>4.186</td>
<td>0.27</td>
<td>25.829</td>
<td>0.07</td>
<td>28.848</td>
<td>0.09</td>
<td>30.591</td>
<td>0.04</td>
</tr>
<tr>
<td>0.8</td>
<td>2</td>
<td>0.2</td>
<td>0.2</td>
<td>4.275</td>
<td>0.25</td>
<td>23.979</td>
<td>0.12</td>
<td>28.648</td>
<td>0.09</td>
<td>33.809</td>
<td>0.05</td>
</tr>
<tr>
<td>50</td>
<td>2</td>
<td>0.2</td>
<td>0.2</td>
<td>15.142</td>
<td>0.28</td>
<td>52.230</td>
<td>0.08</td>
<td>55.552</td>
<td>0.09</td>
<td>54.340</td>
<td>0.04</td>
</tr>
<tr>
<td>0.5</td>
<td>2</td>
<td>0.2</td>
<td>0.2</td>
<td>18.500</td>
<td>0.31</td>
<td>56.218</td>
<td>0.17</td>
<td>60.054</td>
<td>0.21</td>
<td>63.509</td>
<td>0.02</td>
</tr>
<tr>
<td>0.5</td>
<td>2</td>
<td>0.2</td>
<td>0.2</td>
<td>21.852</td>
<td>0.24</td>
<td>64.885</td>
<td>0.15</td>
<td>67.550</td>
<td>0.12</td>
<td>72.884</td>
<td>0.05</td>
</tr>
<tr>
<td>0.5</td>
<td>2</td>
<td>0.2</td>
<td>0.2</td>
<td>15.984</td>
<td>0.17</td>
<td>63.104</td>
<td>0.08</td>
<td>63.409</td>
<td>0.09</td>
<td>65.751</td>
<td>0.04</td>
</tr>
<tr>
<td>0.5</td>
<td>2</td>
<td>0.2</td>
<td>0.2</td>
<td>20.871</td>
<td>0.26</td>
<td>50.690</td>
<td>0.06</td>
<td>58.900</td>
<td>0.11</td>
<td>66.529</td>
<td>0.00</td>
</tr>
<tr>
<td>Mean</td>
<td></td>
<td></td>
<td>12.166</td>
<td>0.24</td>
<td>39.895</td>
<td>0.09</td>
<td>51.683</td>
<td>0.11</td>
<td>45.183</td>
<td>0.11</td>
<td>46.243</td>
</tr>
<tr>
<td>3</td>
<td>20</td>
<td>0.2</td>
<td>0.2</td>
<td>21.763</td>
<td>0.24</td>
<td>49.312</td>
<td>0.15</td>
<td>59.283</td>
<td>0.12</td>
<td>53.730</td>
<td>0.07</td>
</tr>
<tr>
<td>0.5</td>
<td>2</td>
<td>0.2</td>
<td>0.2</td>
<td>23.804</td>
<td>0.29</td>
<td>54.602</td>
<td>0.13</td>
<td>65.935</td>
<td>0.14</td>
<td>64.548</td>
<td>0.00</td>
</tr>
<tr>
<td>0.5</td>
<td>2</td>
<td>0.2</td>
<td>0.2</td>
<td>21.858</td>
<td>0.24</td>
<td>64.885</td>
<td>0.15</td>
<td>67.550</td>
<td>0.12</td>
<td>72.884</td>
<td>0.05</td>
</tr>
<tr>
<td>0.5</td>
<td>2</td>
<td>0.2</td>
<td>0.2</td>
<td>15.984</td>
<td>0.17</td>
<td>63.104</td>
<td>0.08</td>
<td>63.409</td>
<td>0.09</td>
<td>65.751</td>
<td>0.04</td>
</tr>
<tr>
<td>0.5</td>
<td>2</td>
<td>0.2</td>
<td>0.2</td>
<td>20.871</td>
<td>0.26</td>
<td>50.690</td>
<td>0.06</td>
<td>58.900</td>
<td>0.11</td>
<td>66.529</td>
<td>0.00</td>
</tr>
<tr>
<td>Mean</td>
<td></td>
<td></td>
<td>30.600</td>
<td>0.27</td>
<td>68.252</td>
<td>0.14</td>
<td>75.545</td>
<td>0.13</td>
<td>75.550</td>
<td>0.03</td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>20</td>
<td>0.2</td>
<td>0.2</td>
<td>43.532</td>
<td>0.27</td>
<td>71.237</td>
<td>0.22</td>
<td>73.550</td>
<td>0.15</td>
<td>76.063</td>
<td>0.00</td>
</tr>
<tr>
<td>0.5</td>
<td>2</td>
<td>0.2</td>
<td>0.2</td>
<td>39.829</td>
<td>0.26</td>
<td>64.494</td>
<td>0.14</td>
<td>73.116</td>
<td>0.17</td>
<td>73.861</td>
<td>0.04</td>
</tr>
<tr>
<td>0.5</td>
<td>2</td>
<td>0.2</td>
<td>0.2</td>
<td>38.293</td>
<td>0.29</td>
<td>83.368</td>
<td>0.15</td>
<td>89.607</td>
<td>0.17</td>
<td>90.041</td>
<td>0.08</td>
</tr>
<tr>
<td>0.5</td>
<td>2</td>
<td>0.2</td>
<td>0.2</td>
<td>35.520</td>
<td>0.25</td>
<td>89.819</td>
<td>0.11</td>
<td>105.351</td>
<td>0.12</td>
<td>96.698</td>
<td>0.06</td>
</tr>
<tr>
<td>0.5</td>
<td>2</td>
<td>0.2</td>
<td>0.2</td>
<td>39.053</td>
<td>0.27</td>
<td>74.982</td>
<td>0.18</td>
<td>91.405</td>
<td>0.14</td>
<td>85.951</td>
<td>0.02</td>
</tr>
<tr>
<td>0.5</td>
<td>2</td>
<td>0.2</td>
<td>0.2</td>
<td>32.049</td>
<td>0.28</td>
<td>77.101</td>
<td>0.15</td>
<td>84.607</td>
<td>0.12</td>
<td>78.391</td>
<td>0.07</td>
</tr>
<tr>
<td>Mean</td>
<td></td>
<td></td>
<td>53.799</td>
<td>0.32</td>
<td>88.358</td>
<td>0.16</td>
<td>96.664</td>
<td>0.14</td>
<td>93.677</td>
<td>0.06</td>
<td></td>
</tr>
<tr>
<td>Mean of all</td>
<td></td>
<td></td>
<td>32.188</td>
<td>0.27</td>
<td>65.502</td>
<td>0.13</td>
<td>72.464</td>
<td>0.13</td>
<td>71.823</td>
<td>0.04</td>
<td></td>
</tr>
</tbody>
</table>
completely comparable consequences, since they have at most 4% deviation in converging the optimal solution in average. As it could be seen, for two-machine category, both of them have 9% gap with the optimum solution in average. Another issue is the average spent computational time via IWD which is more than GA. This matter could be owing to being constructive of IWD algorithm with respect to GA which is an evolutionary one.

Now, we need to check whether the differences observed in Table 3 are statistically significant or not. Accordingly, we perform analysis of variance (ANOVA) over the obtained results. The response variable is the PRE and the controlled factors are $M$, $N$, $\upsilon$ and $\rho$. Table 4 shows the ANOVA output for the obtained PRE of all algorithms.

As it could be observed, since the $p$-value is zero, the differences among all algorithms (Table 3) are significant and consequently the Tukey’s test in 95% confidence interval for the PREs is conducted. Figure 10 demonstrates the responding means plot and honest significant difference (HSD) intervals at 95% confidence level. Such a figures are overall grand pictures. It should be remembered that overlapping intervals specifies that there is no statistically significant difference among the overlapped means (Naderi and Ruiz 2010).

As it could be seen, there’s obviously no significant difference between GA and IWD, however the proposed HIWD has significant difference with the other applied algorithms and yields the best consequence.

In medium-to-large-size instances, the obtained RPDs demonstrate that the HIWD again outperforms all GA, SA and IWD algorithms. It should be mentioned that by increasing the size of sample instances, the proposed HIWD spends lesser computational time in average rather than IWD. This issue is due to employing VNS as an efficient local search within HIWD which shows the VNS explores the search space effectively. In this category of sample generated instances, GA and IWD have again thoroughly comparable performances. Despite the GA outperforms IWD in two-machine category, however, they yield the same RPD in average.

Again, we need to check whether the differences observed in Table 5 are statistically significant or not. To do so, ANOVA is applied over the results. The response variable is the RPD and the controlled factors are $M$, $N$, $\upsilon$ and $\rho$. Table 6 demonstrates the ANOVA output for the obtained RPD of all algorithms.

Since the differences are again significant ($p$-value $= 0.00 < \alpha = 0.05$), the Tukey’s test in 95% confidence interval for the RPDs is conducted. Figure 11 shows the responding means plot and HSD intervals at 95% confidence level.
According to Figure 11, despite GA and IWD do not have significant differences, however, the HIWD statistically outperforms the other employed algorithms. Also, SA has the worst consequences among all algorithms in both small- and medium-to-large-sized problems.

Figure 12 demonstrates the behaviour of pure IWD and HIWD in converging to optimal solution for a given problem. As it could be seen, the convergence is attained in HIWD in a stronger manner compared to the pure IWD, since HIWD benefits a capable local search, i.e. VNS. In fact, the VNS capability in exploring the search space helps IWD to converge to optimum solutions.

Generally speaking, according to the obtained results and statistical tests, one could say that the proposed HIWD is a proficient algorithm which well benefits the useful features of both IWD and VNS algorithms, especially in large-size problems.

7. Conclusions and future works

In this study, a promising optimisation technique called IWDs algorithm was presented and applied to solve jobs scheduling problem on parallel machines environment. The objective was to minimise earliness and tardiness as well as maximum completion time, called makespan simultaneously on unrelated parallel machines. The due dates are distinct and jobs are SDSTs. Since the machines are unrelated, jobs processing time/cost on different machines may vary, i.e. each job could be processed at different processing times with regard to other machines. A Mixed Integer Programming (MIP) model was proposed in this study to solve the considered problem.

The IWD algorithm is inspired from natural rivers. A set of good paths among plenty of possible paths could be found via a natural river in its ways from the starting place (source) to the destination which results in eventually finding a high quality path to their destination. In this study, in addition to present the mathematical model as well as the population-based constructive algorithm, IWD, an effective hybrid technique, i.e. HIWD was then proposed by hybridising IWD with VNS method which has the improvement ability.

Two sets of instances in terms of small- and medium-to-large-sized ones, with different number of machines, jobs, set-ups and processing times were generated to evaluate the performance of employed algorithms. In small-size cases, which optimal solution could be found by Lingo, PRE as performance measure was employed in all algorithms while RPD was used for medium-to-large-size instances. ANOVA was carried out over the obtained results of both PREs and RPDs. Since the differences among algorithms were significant in both small- and medium-to-large-size problems, Tukey’s test was conducted in 95% confidence interval. Experimental results demonstrated that the proposed hybrid HIWD algorithm has a much better performance than all other employed ones. This fact reveals that the HIWD is a proficient and capable algorithm to find high quality solutions, especially in large-size cases.

As a direction for future research, it would be interesting to extend the IWD algorithm on other scheduling environments, such as flow-shop or job-shop. Another interesting area may be to implement this algorithm on multi-objective problems which are so attractive and closer to the real-life situations.
References


Duan, H., S. Liu, and X. Lei. 2008. “Air Robot Path Planning based on Intelligent Water Drops Optimization.” In IEEE International Joint Conference on Neural Networks, 2008. IJCNN 2008 (IEEE World Congress on Computational Intelligence), Hong Kong, 1397–1401.


Appendix 1. Linearisation of the proposed mathematical model

In this section, an attempt is made to linearise the mathematical model proposed in this research.

A.1. Procedure

The linearisation procedure which is proposed here consists of two steps which are given by the two propositions stated below. The non-linear terms in the constraint (9) is multiplication of binary and integer variables which could be linearised using the following auxiliary variables \( F_{ikjm} \) and \( G_{ikjm} \). Each proposition for linearisation is followed by a proof that illustrates the meaning of each auxiliary (linearisation) variable and the expressions where they are used.

**Proposition 1.** The non-linear terms in the constraint (9) of the mathematical model could be linearised with \( F_{ikjm} = y_{kj} - 1 \cdot m \cdot y_{ijm} \), under the following sets of constraints:

\[
F_{ikjm} \geq y_{kj-1}m + y_{ijm} - 1.5 \quad \forall j \geq 2, i \neq k, m; \tag{A.1}
\]

\[
1.5 \times F_{ikjm} \leq y_{kj-1}m + y_{ijm} \quad \forall j \geq 2, i \neq k, m; \tag{A.2}
\]

**Proof.** Consider the following two cases:

1. \( y_{kj-1}m \cdot y_{ijm} = 1 \). \( \forall j \geq 2, i \neq k, m; \)

Such a situation arises when \( y_{kj-1}m = y_{ijm} = 1 \). So, constraint (A.1) implies \( F_{ikjm} \geq 0.5 \) which makes sure that \( F_{ikjm} = 1 \).

1. \( y_{kj-1}m \cdot y_{ijm} = 0 \). Such a situation arises under one of the following three cases:
   - (a) \( y_{kj-1}m = 1 \) and \( y_{ijm} = 0 \). \( \forall j \geq 2, i \neq k, m; \)
   - (b) \( y_{kj-1}m = 0 \) and \( y_{ijm} = 1 \). \( \forall j \geq 2, i \neq k, m; \)
   - (c) \( y_{kj-1}m = 0 \) and \( y_{ijm} = 0 \). \( \forall j \geq 2, i \neq k, m; \)

In all of these cases, \( F_{ikjm} = 0 \), because in these cases, constraint (A.2) implies \( 1.5 \times F_{ikjm} \leq 0 \) or 1 and so ensures that \( F_{ikjm} = 0 \). Since \( F_{ikjm} \) does not have a strictly positive cost coefficient, the minimising objective function does not ensure that \( F_{ikjm} = 0 \). Consequently, constraint (A.2) should be added to the mathematical model.

**Proposition 2.** The non-linear constraint (9) could be linearised by the following transformation \( G_{ikjm} = F_{ikjm} \cdot C_i \), under the following sets of constraints:

\[
G_{ikjm} \leq C_i + A(1 - F_{ikjm}) \quad \forall j \geq 2, i \neq k, m; \tag{A.3}
\]

\[
G_{ikjm} \geq C_i - A(1 - F_{ikjm}) \quad \forall j \geq 2, i \neq k, m; \tag{A.4}
\]

\[
G_{ikjm} \leq A \cdot F_{ikjm} \quad \forall j \geq 2, i \neq k, m; \tag{A.5}
\]

**Proof.** Consider the following two sections:

This section could be shown for each of the two possible cases that can arise.

1. \( F_{ikjm} \cdot C_i = C_i \) \( \forall j \geq 2, i \neq k, m; \)

Such a situation arises when \( F_{ikjm} = 1 \). So, constraints (A.3) and (A.4) implies \( G_{ikjm} \leq C_i \) and \( G_{ikjm} \geq C_i \) and ensures that \( G_{ikjm} = C_i \).

1. \( F_{ikjm} \cdot C_i = 0 \). Such a situation arises under one of the following three sub-cases:
   - (a) \( F_{ikjm} = 1 \) and \( C_i = 0 \). \( \forall j \geq 2, i \neq k, m; \)
   - (b) \( F_{ikjm} = 0 \) and \( C_i > 0 \). \( \forall j \geq 2, i \neq k, m; \)
   - (c) \( F_{ikjm} = 0 \) and \( C_i = 0 \). \( \forall j \geq 2, i \neq k, m; \)
In all of the three sub-cases given above, $G_{ikjm}$ takes the value of 0, because in these cases, constraint (A.5) implies $G_{ikjm} \leq 0$ and ensures that $G_{ikjm} = 0$. Since $G_{ikjm}$ does not have a strictly positive cost coefficient, the minimising objective function does not ensure that $G_{ikjm} = 0$. Accordingly, constraint (A.5) should be added to the mathematical model.

A.2. The linearised model
The linear mathematical model is now presented as follows:

$$\text{Min } Z = \text{Eq.(1)}$$

Subject to constraints (2)–(8), (10)–(12) and (A.1)–(A.6).

$$G_{ikjm} \geq 0 \text{ and } F_{ikjm} \in \{0, 1\} \quad \forall j \geq 2, i \neq k, m; \quad (A.6)$$