Parallel Robot Scheduling to Minimize Mean Tardiness with Unequal Release Date and Precedence Constraints Using a Hybrid Intelligent System

Regular Paper

Turk Çakar¹, Rasit Köker²,* and Yavuz Sarı³

¹ Sakarya University Engineering Faculty, Industrial Engineering Department, Sakarya, Turkey
² Sakarya University Technical Education Faculty, Department of Electronics and Computer Sciences, Sakarya, Turkey
³ Sakarya University Hendek Vocational High School, Electronics and Automation Department, Sakarya, Turkey

* Corresponding author E-mail: rkoker@sakarya.edu.tr

Received 9 May 2012; Accepted 15 Oct 2012

DOI: 10.5772/54381

© 2012 Çakar et al.; licensee InTech. This is an open access article distributed under the terms of the Creative Commons Attribution License (http://creativecommons.org/licenses/by/3.0), which permits unrestricted use, distribution, and reproduction in any medium, provided the original work is properly cited.

Abstract This paper considers the problem of scheduling a given number of jobs on a specified number of identical parallel robots with unequal release dates and precedence constraints in order to minimize mean tardiness. This problem is strongly NP-hard. The author proposes a hybrid intelligent solution system, which uses Genetic Algorithms and Simulated Annealing (GA+SA). A genetic algorithm, as is well known, is an efficient tool for the solution of combinatorial optimization problems. Solutions for problems of different scales are found using genetic algorithms, simulated annealing and a Hybrid Intelligent Solution System (HISS). Computational results of empirical experiments show that the Hybrid Intelligent Solution System (HISS) is successful with regards to solution quality and computational time.

Keywords Parallel Robot Scheduling, Genetic Algorithms, Simulated Annealing, Unequal Release Date, Precedence Constraints, Mean Tardiness, Hybrid Intelligent System

1. Introduction

Due to the growing cost of raw materials, labour, energy and an increasingly competitive environment, manufacturers must produce cheaper and high quality products. This results in the need for automation techniques. Robots are one of the most important devices used in the automation of industry. Many industrial automation systems use more than one robot for a task. Job scheduling is necessary to increase the efficiency of the system design. The main aims of parallel robot scheduling are total completion time, maximum earliness and minimal tardiness. From a theoretical perspective, parallel robot scheduling is an extrapolation of single robot scheduling and a particular study of flow shop. From a practical perspective, solution techniques are useful for real-world problems. In practical terms, parallel robot scheduling is faced with having to balance the work load. Scheduling parallel robots may be seen as a two-step process. The first step is deciding which jobs are
allocated to which robot and secondly, allocating the sequence in which the jobs are carried out. Also, preemption plays a more important role in parallel robot scheduling. Robots may be identical or not. Jobs have precedence constraints [1–2]. Many manufacturing systems have parallel robot architectures. An industrial object classification system with two robots can be used as a case study for the scheduling application. Another example can be the reduction of assembly time in the assembly of printed circuit boards (PCB) by prioritizing the efficient simultaneous pickup operation of placement robots. The scheduling increases the efficiency of designed systems [3].

The problems of parallel machine scheduling have been extensively studied in the literature [4–8]. State-of-the-art reviews on research of parallel machines have been presented by Graves et al. (1982) [5] and by Mokotoff (2001) [7]. Additionally, Allahverdi et al. (1999) published a survey paper about the scheduling problems involved in setup time constraints [6]. Picard and Queyranne (1977) [4] and Brucker et al. (1998) [9], Allahverdi et al. (1999) [6], Asano and Ohta (2002) [8] developed branch and bound algorithms for these types of problems.

Although the focus of our review will be limited to the unrelated Parallel Machine Scheduling Problem (PMSP), it is important to mention that many papers address identical parallel machine scheduling with and without setup consideration; these include recent studies by Dunstall and Wirth (2005) [10], Kurz and Askin (2001) [11] and Lin and Li (2004) [12]. Kim and Bobrowski (1997) used combined neural networks with some dispatching rules in their study [13], Kim et al. (2002) [14] and Kim et al. (2003) [15] focused on developing heuristics for the problem, which includes a Simulated Annealing (SA) for the minimization of the total tardiness with machine-independent sequence-dependent setup times.

Glass et al. (1994) presented a study based on the comparison of algorithms (GA), SA and Tabu Search (TS) for Rm|Cmax without setup times [16]. In their paper, they concluded that the quality of solutions obtained with GA was poor. However, a hybrid method including GA was found to be comparable to the performance of SA and TS.

Li and Yang (2011) consider the uniform parallel machine scheduling problem with unequal release dates and delivery times with the aim of minimizing the maximum completion time [17]. Jouglet and Savourey (2011) address the parallel machine total weighted tardiness scheduling problem with release dates. We describe dominance rules and filtering methods for this problem [18].

Saricicek and Celik (2011) focus on the problem of scheduling n independent jobs to be processed on m identical parallel machines with the aim of minimizing the total tardiness of the jobs with a job splitting property [19]. Vallada and Ruiz (2011) use a genetic algorithm for the unrelated parallel machine scheduling problem, in which machine and job sequence dependent setup times are considered [20]. Lin et al., (2011) compare the performance of various heuristics and one metaheuristic for unrelated parallel machine scheduling problems [21]. The objective functions to be minimized are makespan, total weighted completion time and total weighted tardiness. Balin (2011) proposes a new “crossover operator” and a new “optimality criterion” in order to adapt the GA to non-identical parallel machine scheduling problems [22]. Lin et al. (2011) solved an unrelated parallel machine scheduling problem with sequence- and machine-dependent setup times in the presence of due date constraints [23]. The paper by Li, Yang and Ma, (2011) considers the uniform parallel machine scheduling problem of minimizing the maximum lateness [24]. Chang and Chen (2011) deal with an unrelated parallel machine scheduling problem with the objective of minimizing the makespan [25]. Toksari and Guner (2010) focus on analyzing a parallel machine earliness/tardiness (ET) scheduling problem with the simultaneous effects of learning and linear deterioration, sequence-dependent setups and a common due-date for all jobs [26]. Biskup et al. (2008) considered the problem of scheduling a given number of jobs on a specified number of identical parallel machines to minimize total tardiness [27]. In order to solve a type of identical and non-identical parallel machine scheduling problem in order to minimize the total weighted completion time, a genetic algorithm with a new extended representation encoding was proposed by Zhou et al. (2007) [28]. Gomez-Gasquet et al. present an experimental investigation on unrelated parallel machine scheduling using dispatching rules [29]. On the other hand, exact algorithms for unrelated PMSP have been developed by some researchers including Liaw et al. (2003) [30]. Missbauer (1997) presented an investigation into order release and sequence-dependent setup times [31]. Because of the complexity of the problem, finding optimal solutions for large problems is very time consuming and sometimes computationally unfeasible. Developing heuristic algorithms to derive near-optimal solutions becomes much more practical and more useful. Using meta-heuristic approaches for related PMSP has been considered by some authors.

A study based on effective TS for the same problem without setup times was presented by Srivastava (1998) [32]. In his paper, as a conclusion, the solutions obtained from TS are found to be applicable for practical-sized problems within a reasonable amount of time. Martello et al. (1997) presented a study based on a mixed integer programming model and heuristic algorithms [33]. In Chen and Powell (1999), a column generation strategy is
proposed [34]. Salem et al. (2000) solve the problem with a tree search method [35]. Néron et al. (2008) compare two different branching schemes and several tree search strategies for the problem with release dates and tails for the makespan minimization case [36]. Baev et al. (2002) and van den Akker et al. (2005) deal with the problem by using precedence constraints for the minimization of the sum of completion times and maximum lateness, respectively [37-38].

Kashara and Narita (1985) developed a heuristic algorithm and an optimization algorithm for the parallel processing of robot arm control computation on a multiprocessor system [39]. Chen et al. (1988) developed a state-space search algorithm coupled with a heuristic for robot inverse dynamics computation on a multiprocessor system [40]. An assignment rule, labelled traffic priority index (TPI), was built by Ho and Chang (1991) [41]. In this method, SPT and EDD rules are combined by using a new measurement named traffic congestion ratio (TCR). Then, for cases with one or identical machines they built heuristics. Their heuristics consist of building a first solution by scheduling jobs in increasing order of their priority index. Then they improved this solution using the permutation technique of the WI method, which was developed previously by Wilkerson and Irwin (1971) [42]. Cakar et al. (2008) proposed a genetic algorithm approach to solve parallel robot scheduling with precedence constraints [43]. Kanjo and Ase (2003) studied scheduling in a multi robot welding system [44]. Jun and Ying (2002) applied a genetic algorithm for the scheduling of dual resources with robots [45]. Zacharia and Asparagatos (2005) proposed a method for GAs for optimal robot task scheduling [46]. In this study, the job with n-number of precedence constraints and unequal release dates is tasked with minimizing mean tardiness on m-number of parallel robots using genetic algorithms.

This paper deals with the problem of scheduling a given number of jobs on a specified number of identical parallel robots with unequal release dates and precedence constraints so as to minimize mean tardiness. Figure 1, is a symbolic representation of parallel working robots. A hybrid intelligent solution system, which uses Genetic Algorithms and Simulated Annealing is proposed. The solutions of problems in different scales were carried out using genetic algorithms, simulated annealing and a Hybrid Intelligent Solution System (HISS). In this paper, the second section discusses modelling the target function; the third section looks at modelling the problem using genetic algorithms (GA); the fourth section looks at modelling the problem using simulated annealing (SA); the fifth section compares GA and SA; and in the sixth section, the Hybrid Intelligent Solution System (HISS) is described. In the seventh section, an example problem is solved and the results are compared. In the eighth section, the experimental design is shown. And finally, in the ninth section, the conclusions are given.

Figure 1. Symbolic representation of parallel working robots

2. Formulation of the objective function

In this study, a job with n-number of precedence constraints and unequal release dates is scheduled, minimizing mean tardiness on m-number of parallel robots. There is a process time, release date and due date for each job. A robot can do just one job at a time. The processing is non-pre-emptive. The target function to be minimized is given below in Eq. (1).

\[
\text{MeanTardiness} = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} R(i,j)T_{j}}{n}
\]

Here, \(T_{i} = \max [0, C_{j} - d_{j}]\) is the tardiness of job \(j\). \(C_{j}\) being the completion time and \(d_{j}\) being the due date for the job \(j\). \(R(i,j)\) represents the processing or unprocessing of job \(j\) on robot \(i\). If job \(j\) is being processed on robot \(i\), \(R(i,j)=1\), otherwise (if not being processed) \(R(i,j)=0\).

3. Genetic algorithms

In this section, the modelling and the application of the GA are explained. As regards the working principle, a genetic algorithm firstly needs the coding of the problem so long as it fits with the GA. After the coding process, GA operators are applied to chromosomes. The working of crossover and mutation operators does not guarantee that the obtained new offspring are good solutions. Feasible solutions are evaluated and others are left out of the evaluation. The feasible solutions from the offspring, thus obtained, are taken and new populations are formed by a reproduction process using these offspring. Crossover, mutation and reproduction processes go on until an optimal solution is found. The modelling of the defined problem using a genetic algorithm is presented below with the relevant details.

3.1 Modelling of the problem using Genetic Algorithms

The scheduling of the jobs on each robot forms the chromosomes. Here, the chromosomes give the number of robots too. The gene codes are \(c_{i}, c_{2}, c_{3}, \ldots, c_{m}\), where \(c_{i} \in [1,m]\). \(c_{i}\) is positive integer number. Here, each parallel robot represents a chromosome, and the gene in the chromosome represents ordered jobs on a robot. The
assignment of jobs to robots when forming the initial population is done at random and while this ordering is done, precedence constraints are taken into consideration. For instance, let us suppose that there are 9 jobs and 3 robots and their precedence constraints are given in Figure 2. A sample list representing the schedule of the jobs on robots M1, M2 and M3 is given in Figure 3. The sample schedule also gives a sample gene code. In Figure 3, r refers to release date and is shown in the figure as r1, r2, ... and r9 designating the release date of first, second ... and ninth job respectively.

Gene code of the feasible schedule:

M1: J1-J2-J3
M2: J4-J5-J7
M3: J6-J8-J9

3.2 Preparing Initial Population

The initial population is not produced completely at random. In the initial population, the solutions, which are obtained from SPT, LPT, CR and EDD priority rules, exist. The chromosomes out of these are generated randomly. The jobs are randomly assigned (determined or given) to robots. However, because of the precedence constraints - in other words, there are some situations where some jobs may be done before others - some of the solutions obtained will not be feasible. These solutions will be thrown out and the new solutions will be tried randomly.

3.3 Crossover

The chromosomes were set up in a partial structure at the beginning since the jobs are done in the parallel machines. Later on, these parts were joined together and a unique chromosome structure was obtained. After that, the crossover operation was carried out based on this unique chromosome structure.

First, a linear crossover is applied to these chromosomes and new offspring are obtained. However, these offspring cannot be an alternative solution in their existing cases; it is necessary to cut these offspring at randomly decided points to obtain partial chromosomes. Firstly, the chromosome is divided to ensure that each machine has an equal amount of jobs. In our example given below, since
there are three parallel machines, offspring should be cut at two points to produce chromosomes – in other words, to produce a job schedule for each machine. Since this cutting process will be done randomly, it is useful to repeat this process a few times to produce solution alternatives. The repetition process is decided based on one third of the number of genes in one chromosome. For example, if there are 9 genes in the chromosome, 9/3 = 3 repetitions will be done. Thus, 4 alternative solutions are obtained in total. In these alternatives, if there is any solution which does not fit with the precedence constraints, or in other words, an unfeasible solution is killed, other surviving solutions are kept so as to proceed with the reproduction operation.

The Linear Order Crossover (LOX) method is applied to each chromosome independently. LOX works as given below:

1. Select the sublist from chromosomes randomly:
   chromosome #1: 147258369
   chromosome #2: 123456789

2. Remove the sublist 2 from chromosome #1:
   chromosome #1: 1h72h83h9
   chromosome #1: 172hh83h9
   offspring #1: 172456839

3. Remove the sublist 1 from chromosome #2:
   chromosome #2: 1h34h67h9
   chromosome #2: 134hh67h9
   offspring #2: 134258679

4. Insert sublists into holes to form offspring:
   offspring #1: 172456839
   offspring #2: 134258679

In the example, the number of machines was given as 3. In the newly obtained chromosome each gene represents one job. Therefore, there are 9 jobs. 9/3 = 3, 3 jobs are assigned to each machine starting from the first machine, sequentially.

<table>
<thead>
<tr>
<th>CHROMOSOME</th>
<th>MUTATION</th>
<th>OFFSPRING</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1 1 7 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALTERNATIVE #1</td>
<td>M2 4 5 6</td>
<td></td>
</tr>
<tr>
<td>M3 8 3 9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If we focus on random points to be cut, each chromosome has 9 genes, so it means 8 cutting points. If we get two cutting points between 1 and 8 as 2 and 7, it will be cut as shown below:

17 | 24568 | 39

<table>
<thead>
<tr>
<th>CHROMOSOME</th>
<th>MUTATION</th>
<th>OFFSPRING</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1 1 7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALTERNATIVE #2</td>
<td>M2 4 5 6 8</td>
<td></td>
</tr>
<tr>
<td>M3 3 9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the second experiment, these random cutting points are randomly chosen as 4 and 7, giving the following:

1724 | 568 | 39

<table>
<thead>
<tr>
<th>CHROMOSOME</th>
<th>MUTATION</th>
<th>OFFSPRING</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1 1 7 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALTERNATIVE #3</td>
<td>M2 5 6 8</td>
<td></td>
</tr>
<tr>
<td>M3 3 9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

In the third experiment, these random cutting points are chosen as 3 and 8 and then it will be as shown below:

172 | 45683 | 9

<table>
<thead>
<tr>
<th>CHROMOSOME</th>
<th>MUTATION</th>
<th>OFFSPRING</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1 1 7 2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ALTERNATIVE #4</td>
<td>M2 4 5 6 8 3</td>
<td></td>
</tr>
<tr>
<td>M3 9</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

If we look at the precedence constraints, we will see that the first and second alternatives are feasible and the third and fourth alternatives are not feasible. The unfeasible solution alternative will be killed and others will survive to be used in the reproduction process.

3.4 Mutation

During the mutation process, two genes are selected randomly and their places are exchanged to obtain a new offspring. The obtained offspring can be a feasible solution or an unfeasible solution. In the example below, job number 4 in the second row of the first machine and job number 2 in the first row of the second machine were randomly selected. The places of these two jobs were exchanged and a new offspring; or in other words a new alternative solution, was obtained. It can be seen that the obtained offspring is a feasible solution.

<table>
<thead>
<tr>
<th>CHROMOSOME</th>
<th>MUTATION</th>
<th>OFFSPRING</th>
</tr>
</thead>
<tbody>
<tr>
<td>M1 1 4 7 9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>M2 2 5 8</td>
<td></td>
<td>M2 4 5 8</td>
</tr>
<tr>
<td>M3 3 6</td>
<td></td>
<td>M3 3 6</td>
</tr>
</tbody>
</table>

3.5 Reproduction

A copy of each gene is made in the population by the reproduction operator and added to the list of candidate genes. Basically, this guarantees that each chromosome in the current population remains a candidate to be selected for the next population. In this problem, the goal is to find the solution, which minimizes the given fitness function. As we already know, the fitness function is a tardiness value function. Here, the obtained chromosomes are scheduled from low tardiness value to high tardiness value in every population. GA may have better chances at finding better solution results with the surviving chromosomes with quite higher fitness. The living good chromosomes stay in the population, while
others are killed. This process will continue until an optimal solution is found in each population.

4. Simulated annealing

In this study, two operators were used in the application of SA. The first operator is the swapping of a randomly selected job with another job on the same level and then, a new offspring is obtained. The second operator is, again, the swapping of a randomly selected job with another job and then a new alternative solution is obtained. If these obtained alternative solutions are valid, they are taken into consideration.

SA begins with an initial solution (A), an initial temperature (B) and an iteration number (C). The duty of temperature (T) is controlling for the possibility of the acceptance of a disturbing solution and an iteration number (C) is used in deciding the number of repetitions needed until a solution has a stable state according to the temperature. T may have the following implicit meaning as regards the flexibility index: in a high temperature situation – in other words, early in the search - there is some flexibility allowing movement towards a worse solution situation. At a lower temperature, namely later in the search, less of this flexibility exists. A new neighbourhood solution (N) is generated based on these B, C through a heuristic perturbation of the existing solutions. If the changing of an objective function is improved, the neighbourhood solution (N) becomes a good solution. Although it is not improved, the neighbourhood solution will be a new solution with a convenient probability that is based on $e^{-\lambda T}$. This situation creates the possibility of finding a global optimal solution out of a local optimum. The algorithm will be stopped when there is no change after C iterations. Otherwise, the algorithm will continue with a new temperature value (T).

4.1 Simulated annealing algorithm

Begin;
INITIALIZE (A,B,C);
Repeat
For I=1 to C do
    N= PERTURB (A); [generate new neighbourhood solution]
    D= C(N)-C(A)
    If((C(N)<C(A) or (exp(-D/T)>RANDOM(0,1))
    Then A=N; [Accept the movement]
Endif
Endfor;
UPDATE (T, C);
Until (Stop-Criterion)
End;

In order to use SA for practical problems, there will be several factors to be decided upon initially. The first is the definition of a procedure for the generation of neighbourhood solutions from a current solution. For the efficient generation of these solutions, some parameters should be decided upon appropriately. Some examples of these parameters may be an initial temperature, the number of repetitions, the ratio of temperature change and conditions for completion. To obtain a good solution, the combination of these parameters should be adjusted according to the problem.

SA has some weak points such as a long process time and its difficulty in selecting a cooling parameter in the case of problems of a larger size. A geometric ratio was used in SA of $T_{k+1} = \alpha T_k$ where $T_k$ and $T_{k+1}$ are the temperature values for $k$ and $k+1$ steps, respectively. In practice, a geometric ratio is used more commonly. During this study, the initial temperature was taken as 10000 and 0.95 was used for the cooling ratio ($\alpha$).

OPERATOR-1 works by choosing the jobs on the same hierarchic level and swapping them. If we look at the precedence constraints, it is seen that the jobs are on the same level. The chosen job numbers 7 and 8 were swapped and the obtained solution is feasible.

\[
\begin{array}{ccc}
\text{OLD} & \text{SA OPERATOR-1} & \text{NEW} \\
\text{SOLUTION} & & \\
M1 & 1 & 4 & 7 & 9 & M1 & 1 & 4 & 8 & 9 \\
M2 & 2 & 5 & 6 & \Rightarrow & M2 & 2 & 5 & 7 \\
M3 & 3 & 6 & & & M3 & 3 & 6 & \\
\end{array}
\]

OPERATOR-2 works by swapping two randomly selected jobs. In the example, jobs 1 and 6 are swapped. The solution obtained does not fit with the precedence constraints. In other words, the obtained solution is not feasible.

\[
\begin{array}{ccc}
\text{OLD} & \text{SA OPERATOR-2} & \text{NEW} \\
\text{SOLUTION} & & \\
M1 & 1 & 4 & 7 & 9 & M1 & 6 & 4 & 7 & 9 \\
M2 & 2 & 5 & 8 & \Rightarrow & M2 & 2 & 5 & 8 \\
M3 & 3 & 6 & & & M3 & 3 & 1 & \\
\end{array}
\]

5. Proposed hybrid intelligent solution system

GA and SA do not have too many differences; both of them are theoretically quite pertinent algorithms. However, their formulations are based on using very different terminologies. For instance, in a problem solution with SA, the costs, neighbours and moves of the solutions are discussed; on the other hand, in a problem solution with GA, the discussions are based on chromosomes, their crossover, fitness and mutation. Another difference is that a chromosome is considered as a genotype indicating only a solution. This is a traditional characteristic feature of GA and there is no reason why a similar approach could not be used in SA in the same way. Basically, SA can be considered as
GA for cases where the population size is only one. Because there is only one chromosome and there is not any crossover, but there is only mutation. Actually, this is the most important difference between GA and SA. In the SA, a new solution is generated based on modifying only one solution with a local move; however, in the GA, solutions are generated based on using the different solutions in combination. It is not definitely known whether this actually makes the algorithm better or worse, but it is clear that this depends on the problem and the representation. The principles of GA and SA are based on the same fundamental supposition that suitable solutions are more probably found “near” already known suitable solutions than by randomly selecting from the whole solution space. If this was not the case with a particular problem or representation then they would not perform better than random sampling. The difference in the way the GA works is to treat combinations of two existing solutions as being “near”, assuming that such combinations (children) share significant amounts of the properties of their parents, so that a child of two convenient solutions is more probably a good solution than a random one. This is only valid for a special problem or representation; otherwise GA will not have an advantage over SA. This case should be highlighted as significant. On the other hand, using GA and SA in a hybrid system can help to decrease the computation time.

![Diagram](https://via.placeholder.com/150)

**Figure 5.** The proposed Hybrid Intelligent Solution System

In this paper, a hybrid intelligent system solution model is proposed for parallel robot scheduling to minimize mean tardiness with unequal release date and precedence constraints. In the proposed hybrid solution method, GA and SA work together. SA starts its work by taking the best solution of GA. When SA finds a better solution than the one obtained in GA, SA transfers this solution to GA. This solution is used in the reproduction process together with the offspring produced by GA to form the new population. In the new population, the best solution is transferred to SA as the initial population and SA searches for the better solution again. SA works here as an assistant operator of GA. The proposed hybrid system is faster and finds better solutions. The proposed system is shown schematically in Figure 5.

**6. Example problem**

A parallel robot problem with 10 jobs and 3 parallel robots is considered below. The process times, delivery times and precedence constraints of the jobs were given. The solution, which minimizes mean tardiness, was obtained by considering this data. The problem was solved by using SPT (Shortest processing time), EDD (Earliest due date), FCFS (First come first served), CR (Critical ratio), SA, GA and a Hybrid Intelligent Solution System (GA+SA). The data and the results are given below. In Table 1, the jobs with process and due dates belonging to them are given. The precedence constraints of the jobs are given in Figure 6. In Table 2, the solutions obtained from the Hybrid System, GA, SA, SPT, EDD, FCFS and CR are given.

<table>
<thead>
<tr>
<th>Job</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Processing time</td>
<td>3</td>
<td>2</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>7</td>
<td>4</td>
<td>9</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>Due date</td>
<td>2</td>
<td>1</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>3</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>6</td>
</tr>
<tr>
<td>Release date</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>1</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td>5</td>
<td>4</td>
<td>3</td>
</tr>
</tbody>
</table>

**Table 1.** Processing time, due date and unequal release date of every job for the example problem
The number of jobs used in the problems in this study are given in Table 3. In this table, i denotes the jobs, and pij is an integer: processing time and wi is an integer: weight, which were generated from two uniform distributions. The functions of [1, 10] and [1, 100] are to create low or high variations, respectively. TF, which is the relative range of due dates, RDD and an average tardiness factor, were selected from the set [0.1, 0.3, 0.5, 0.7, 0.9]. Here, di is an integer: due date - from the uniform distribution [P (1-TF-RDD/2), P(1-TF+RDD/2)] and it was generated for each job i. In these expressions, P denotes total processing time. Release dates are generated from a uniform distribution between 0 and μΣ pij. As summarized in Table 3, 80000 example sets were considered in total. All algorithms were coded in C++ and implemented on a Pentium IV 2.4 GHz computer during the implementation of algorithms.

As seen in Figure 7, to obtain an optimal solution, the different population size values were applied. When the population size is selected as 25, the optimal solution obtained is found to be better than ones examined with other population sizes. The population size does not affect the solution when it is higher than 25. This is because some of the results of the crossover and mutation processes are accepted as invalid solutions since they cannot give an appropriate order for precedence constraints during the running of GA. Therefore, the population size, which is higher than 25, does not affect the solution as significantly as shown in Figure 7 and furthermore it increases the process time. Additionally, the genetic algorithm was stopped once it had produced the same solution in 50 consecutive generations.
Figure 7. The obtained optimal solutions according to the different population sizes

Figure 8. The effect of crossover rate to solution

In Figure 8, the effect of crossover rate on the solution is shown. From the figure it can be clearly observed that when the crossover rate increases, better solutions can be obtained in better processing times.

Figure 9. The obtained tardiness values for initial population and generations 50, 100 and 150
In Figure 9, the tardiness values are given for the initial population, generation 50, generation 100 and generation 150. The graphic is created by using the best 25 chromosomes obtained from the initial population, generation 50, generation 100 and generation 150. As can be seen in the figure, a better population was obtained in generation 100.

8. Results and discussions

In this section the results are presented and evaluated based on the problems in different sizes for GA, SA and the designed GA+SA hybrid system.

![Table 4. The results of the problems in different sizes](image)

<table>
<thead>
<tr>
<th>Problem size</th>
<th>Number of examples</th>
<th>Number of examples where GA is better than SA</th>
<th>Number of examples where GA is equal to SA</th>
</tr>
</thead>
<tbody>
<tr>
<td>10 x 2</td>
<td>100</td>
<td>79</td>
<td>21</td>
</tr>
<tr>
<td>20 x 3</td>
<td>100</td>
<td>84</td>
<td>16</td>
</tr>
<tr>
<td>30 x 5</td>
<td>100</td>
<td>86</td>
<td>14</td>
</tr>
<tr>
<td>40 x 6</td>
<td>100</td>
<td>89</td>
<td>11</td>
</tr>
<tr>
<td>50 x 7</td>
<td>100</td>
<td>90</td>
<td>10</td>
</tr>
<tr>
<td>60 x 8</td>
<td>100</td>
<td>91</td>
<td>9</td>
</tr>
<tr>
<td>70 x 8</td>
<td>100</td>
<td>96</td>
<td>4</td>
</tr>
<tr>
<td>80 x 8</td>
<td>100</td>
<td>98</td>
<td>2</td>
</tr>
<tr>
<td>90 x 8</td>
<td>100</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>100 x 8</td>
<td>100</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>120 x 8</td>
<td>100</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>150 x 8</td>
<td>100</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>170 x 8</td>
<td>100</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>200 x 8</td>
<td>100</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>220 x 8</td>
<td>100</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>250 x 8</td>
<td>100</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>300 x 8</td>
<td>100</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>400 x 10</td>
<td>100</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>500 x 10</td>
<td>100</td>
<td>100</td>
<td>0</td>
</tr>
<tr>
<td>600 x 10</td>
<td>100</td>
<td>100</td>
<td>0</td>
</tr>
</tbody>
</table>

According to Table 10, it is evident that the Hybrid Intelligent Solution System reaches the solution in a better process time and it can also find better solutions. In other words, adding SA to GA increased the performance of GA. GA and SA metaheuristics have some advantages compared to each other because of the algorithms which they use. Therefore, these two algorithms can be implemented in parallel and then they can share the best solution with each other to find a faster and better solution. It can clearly be seen in Fig. 10, that the hybrid solution system, which uses both algorithms together, gave better results compared to GA. The hybrid intelligent solution system reached the solution quicker. The hybrid solution system reached the final solution in approximately 50 generations whereas GA reached the final solution in about 92 generations. The hybrid system is more advantageous in terms of process time.

Since the hybrid system reaches the final solution when GA is still trying to reach the final solution, the hybrid system should be stopped from working at this point. In 90x8, GA gave better solutions then SA at a rate of 100% in the sample solutions. GA has shown itself to be successful for dealing with the large size problems. However, we cannot take this success of GA at dealing with the selected problem to apply too generally; in other words, we cannot simply say that GA is more successful then SA. According to the large t-test values for the average improvement, GA provides an important improvement and the amount of improvement is noteworthy at a confidence level of 99.5%.
It can clearly be seen in Table 6 that the hybrid system is more successful than GA. Additionally; the hybrid system is also successful when it comes to the process time, as seen in the table. According to the large t-test values it is evident that hybrid system is more successful than GA at a level of 99.5 %. In Table 7, the number of solutions that are better obtained by using the hybrid system than GA are given. In Table 7, it can be seen that GA and SA played a significant role in increasing their own success in the hybrid intelligent solution system. Especially, as seen in Table 7 the performance of GA has been increased by the hybrid system and more successful solutions have been found.

In conclusion, the results show that the proposed HISS outperforms GA in effectiveness (the quality of the solution) and efficiency (less computational time). In the future, this proposed HISS could be applied to non-identical parallel robots.

10. References


