CORA Cork Open Research Archive
Cartlann Taighde Oscailte Chorcaí

Title	A two-phase hybrid approach for the hybrid flexible flowshop with transportation times
Authors	Armstrong, Eddie;Garraffa, Michele;O'Sullivan, Barry;Simonis, Helmut
Publication date	2022-06-10
Original Citation	Armstrong, E., Garraffa, M., O'Sullivan, B. and Simonis, H. (2022) 'A two-phase hybrid approach for the hybrid flexible flowshop with transportation times', in Schaus, P. (ed.) Integration of Constraint Programming, Artificial Intelligence, and Operations Research. CPAIOR 2022. Lecture Notes in Computer Science, 13292. Springer, Cham. doi: 10.1007/978-3-031-08011-1_1
Type of publication	Article (peer-reviewed)
Link to publisher's version	https://cpaior.org/ - 10.1007/978-3-031-08011-1_1
Rights	© 2022, Springer Nature Switzerland AG. This is a post-peer-review, pre-copyedit version of an article published in Lecture Notes in Computer Science. The final authenticated version is available online at: https://doi.org/10.1007/978-3-031-08011-1_1
Download date	2026-11-05 18:52:22
Item downloaded from	https://hdl.handle.net/10468/13342



UCC
University College Cork, Ireland
Coláiste na hOllscoile Corcaigh

A Two-Phase Hybrid Approach for the Hybrid Flexible Flowshop with Transportation Times

Eddie Armstrong[1], Michele Garraffa[2,3], Barry O'Sullivan[2,3], and Helmut Simonis[2,3]

[1] Johnson & Johnson Research Centre, Limerick, Ireland
[2] Confirm SFI Research Centre for Smart Manufacturing
[3] School of Computer Science, University College Cork, Cork, Ireland

Abstract. We present a two-phase heuristic approach for the Hybrid Flexible Flowshop with Transportation Times (HFFTT) which combines a metaheuristic with constraint programming (CP). In the first phase an adapted version of a state-of-the-art metaheuristic for the Hybrid Flowshop [15] generates an initial solution. In the second phase, a CP approach reoptimizes the solution with respect to the last stages. Although this research is still in progress, the initial computational results are very promising. In fact, we show that the proposed hybrid approach outperforms both the adapted version of [15] and earlier CP approaches.

Keywords: Metaheuristics · Constraint Programming · Scheduling · Hybrid Flowshop

1 Introduction

Real world scheduling problems are generally tackled by means of two different types of approaches. On one hand, mixed integer linear programming (MILP) and constraint programming (CP) approaches put the focus on modelling the scheduling problem, instead of developing a solution algorithm from scratch. This approach shortens the development time and provides a high level of flexibility in the case where the problem formulation needs to be adjusted to take into account new characteristics. In that case, an optimization specialist just needs to adapt the MILP/CP model, then a modern solver will provide high quality solutions by exploiting many years of algorithmic improvements. On the other hand, metaheuristic algorithms have been widely studied, since they can exploit problem-specific properties to perform the search very efficiently. In many cases, this leads to achieving high quality results, at the cost of a higher development time and a reduction in the flexibility/generality of the solution approach.

The combination of both approaches, so called hybrid heuristics, has been receiving significant attention from the research community since the early 2000s (see [2] for a survey about the topic). Most hybrid heuristics rely on efficient MILP solvers, due to their use of powerful mathematical programming techniques. However, the performance of MILP solvers is typically poor in cases

where the model has a weak linear relaxation, e.g. due to the use of big-M variables to represent logical constraints. MILP-based hybrid heuristics are generally known as "matheuristics" [9,6], and they have been widely used for scheduling problems [8,3,5,4]. On the other hand, CP solvers rely on the expressive power of global constraints, and on the effectiveness of propagation algorithms and automatic search heuristics. One of the best known commercial CP solvers, CP Optimizer by IBM [7], offers support for efficiently solving many types of scheduling problems. It provides a model-and-run paradigm, which is quite simple to master and is more generic than the one provided by MILP solvers, since non-linearities and logical constraints can be easily included.

In this paper we propose a hybrid heuristic for a real-world scheduling problem, which combines a metaheuristic with a local search procedure relying on CP Optimizer. Hybrid heuristics based on CP for scheduling problems are not very common, but they have been considered [14,12]. The motivation behind hybridizing CP with another type of solution approach is usually to exploit the complementarity of the two in order to achieve a better performance than each of the two approaches separately. The problem considered in this study is the Hybrid Flexible Flowshop with Transportation Times (HFFTT) which arises in modern production facilities and has been recently introduced [1]. The problem is an extension of the Hybrid Flowshop Problem [13] and of the Hybrid Flexible Flowshop [10] where transportation times between the machines for the different production steps of each job are assumed to be non-negligible.

The HFFTT is defined as follows. Let J be a set of jobs, M a set of machines and S a set of production stages. We denote as $p_{j,s}$ the processing time of a job $j \in J$ to complete a stage $s \in S$. All jobs complete the stages in the same order but some of the stages may be skipped by some jobs. We indicate with $S_j \subseteq S$ the subset of the production stages performed by job $j \in J$, while we indicate with $J_s \subseteq J$ the set of all jobs that complete a stage $s \in S$. The successor stage of a stage $s \in S$ with respect to a job $j \in J$ is denoted as $succ(s, j)$. The transportation time required to move a job $j \in J$ from a machine $m \in M_s$ to another machine $m' \in M_{s'}$, where $s, s' \in S$ and $s' = succ(s, j)$, is represented by $\delta_{m,m'}$. Finally, the problem objective is to minimize the makespan.

The paper is organized as follows. Section 2 describes the CP model of the problem using the global constraints available in CP Optimizer. Section 3 describes an iterated greedy method, `IGT_NEH`, which is an adaptation of a state-of-the-art metaheuristic for the HFP [15]. Section 4 presents a novel two-phase hybrid approach to solve the HFFTT. Section 5 presents a computational assessment of the different approaches, using the benchmarks defined in [1].

2 The Constraint Programming Model

This section presents a CP model of the problem, analogous to the model based on interval variables presented in [1], and was encoded by using the OPL API of CP Optimizer [7]. The model is based on the following variables:

- Optional interval variables $tm_{m,j}$ for each $j \in J$ and $m \in M$;

- Interval variables $ts_{s,j}$ for each $j \in J$ and $s \in S_j$;
- Integer variables $machine_{s,j}$ for each $j \in J$ and $s \in S_j$, with feasible values in the range $\{1, \cdots, |M|\}$.

The variables $tm_{m,j}$ are optional interval variables representing the execution of a job on a certain machine. Given a stage $s \in S$ and a job $j \in J$, only one of these variables is active, which is constrained to be equal to $ts_{s,j}$. The variables $machine_{s,j}$ are linked to the machine used to performed a job $j \in J$ at stage $s \in S$.

The CP model of the HFFTT is as follows:

$$\min \max_{s \in S, j \in J_s} endOf(ts_{s,j}) \tag{1}$$

subject to:

$$(machine_{s,j} = m) \implies presenceOf(tm_{m,j}) \quad \forall j \in J, s \in S_j, m \in M_s \tag{2}$$

$$alternative(ts_{s,j}, \{tm_{m,j} : m \in M_s\}) \quad \forall j \in J, s \in S_j \tag{3}$$

$$endBeforeStart(tm_{m,j}, tm_{m',j}, \delta_{m,m'}) \quad \forall j, s \in S_j, m \in M_s, m' \in M_{succ(s,j)} \tag{4}$$

$$endBeforeStart(ts_{s,j}, ts_{succ(s,j),j})) \quad \forall j \in J, s \in S_j : succ(s,j) \neq \emptyset \tag{5}$$

$$noOverlap(\{tm_{m,j} : j \in J\}) \quad \forall m \in M \tag{6}$$

$$cumulative(\{ts_{s,j} : j \in J_s\}, |M_s|) \quad \forall s \in S \tag{7}$$

The objective (1) indicates that we minimize the makespan. Constraints 2 deal with assigning a job $j \in J$ to one machine $m \in M_s$ at each stage $s \in S_j$, and setting the corresponding interval variable $tm_{m,j}$ to active. Constraints 3 link the interval variables $tm_{m,j}$ and the interval variables $ts_{s,j}$. Constraints 4 indicate that a job $j \in J$ can start being processed by a machine $m' \in M_{succ(s,j)}$ after being completed by the previous one $m \in M_s$ and spending $\delta_{m,m'}$ time units for the transportation. Constraints 5 are flowshop constraints, meaning that each job $j \in J$ can perform the next stage $succ(s,j) \in S_j$ after completing the previous one $s \in S_j$. Constraints 6 state that each machine can process one job at a time. Constraints 7 are redundant, requiring that the maximum number of jobs performing simultaneously at stage $s \in S$ is equal to the number of machines $|M_s|$ available at that stage.

3 Metaheuristic Approach

State-of-the-art metaheuristic approaches for the HFP, e.g. [15], can be easily adapted to solve instances of the HFFTT. These approaches are based on forward scheduling. Given a certain jobs permutation γ, they follow these two steps:

- Assign the jobs by following the order in γ, to the earliest available machine (first stage);
- Assign the earliest available job to the earliest available machine (each of the other stages).

A random choice is taken in case of ties. In such a way, a feasible solution to the HFP can be generated given a reference permutation. The adaptation needed to use forward scheduling on the HFFTT is to consider the transportation times. We do not consider the transportation time when we compute the earliest available job, since the job is available once it has been processed by a certain machine. However, we consider the transportation time when performing the machine assignment. In this case, we do not evaluate the machines according to their earliest available time because the job may not be able to reach the machine at that time. We rank the machines according to the maximum between the:

- Earliest time when the machine is available;
- Earliest time when the job can reach the machine.

This change allows us to replicate approaches based on forward scheduling on the HFFTT.

In the following, we describe our adaptation of the approach denoted as `IGT` (Iterated Greedy with fixed temperature T) [15]. We denote our approach as `IGT_NEH`. The most important changes with respect to `IGT` are:

- `IGT_NEH` considers transportation times when applying forward scheduling;
- `IGT_NEH` computes the initial solution using a different procedure;

Section 3.1, Section 3.2 and Section 3.3 describe the main components of the `IGT_NEH` heuristic, while its structure is discussed in Section 3.4.

3.1 Computation of the Initial Solution

The `NEH` heuristic [11] is one of the most common constructive heuristics used for scheduling problems. It takes its name from the three authors who proposed it for the first time (Nawaz, Enscore and Ham). The approach is quite generic: only the evaluation strategy and the initial sorting criteria change from one problem to another. First, the sum of the processing times on all stages, indicated as $TP_j = \sum_{s \in S} p_{j,s}$, is computed for each $j \in J$. The jobs are then sorted by decreasing order of TP_j, in order to have a good job reference permutation γ. The first job γ_1 of the permutation is selected to establish a partial solution of length one. Then the other jobs in γ are sequentially inserted into the output permutation γ^{out} one by one. At the i-th iteration, the job γ_i is chosen and tentatively inserted into all

the i possible positions of γ^{out}, it is then inserted at the position resulting in the best makespan value. Each evaluation of a permutation is performed by means of forward scheduling and considering the makespan as the objective. Once the n-th iteration is reached, the solution constructed is provided as an output.

We now briefly discuss why we used `NEH` as a method to compute initial solutions, instead of the approach denoted as `GRASP_NEH` in [15]. The computational complexity of `NEH` and `GRASP_NEH` is different ($O(n^3m)$ vs $O(n^4m)$). In our preliminary computational experiments, we explored both variants in the metaheuristic approach. According to our results, spending too much time on computing the initial solution affected the results in large instances ($n \geq 200$), while the two approaches showed quite similar performance in smaller instances. For this reason, we decided to use `NEH` as a method to compute the initial solution.

3.2 Local Search Approaches

The local search moves used in `IGT_NEH` are guided by a reference jobs permutation. They are the same as used in `IGT`, with the only difference that forward scheduling takes into account the transportation times as explained at the beginning of Section 3. These procedures are denoted by `RIS` (Referenced Insertion Scheme) and `RSS` (Referenced Swap Scheme). Given a reference permutation γ and an initial permutation γ^{in} – the job permutation used to generate the initial solution – we iteratively select a job from γ. In `RIS`, we remove the job from γ^{in} and re-insert it in the best position. In `RSS`, we try to swap the selected job with the others in γ^{in} and choose the swap leading to the best objective. The authors in [15] discuss the fact that a reference permutation associated with a high-quality solution can improve the performance of the local search moves. Both local search routines stop whenever n iterations with no improvements are performed. The best solution found is then provided as an output.

3.3 Deconstruction and Reconstruction

The deconstruction-reconstruction procedure, denoted as `DEC_REC`, is straightforward. First, dS random jobs are removed from the initial permutation γ, then they are re-inserted one by one at the best possible position. Again, the only difference with `IGT` is that the transportation times are taken into account when applying forward scheduling.

3.4 General structure of `IGT_NEH`

The `IGT_NEH` procedure is an iterated greedy procedure, which starts from an initial solution and iteratively applies some local search moves, followed by a shaking step to escape local minima. First, an initial solution is computed by using the function `NEH`. Then, a while loop iterates until a time limit $\frac{T_{max}}{2}$ is reached. The loop starts with the deconstruction-reconstruction procedure `DEC_REC`, where dS

jobs are removed from the current permutation and re-inserted to optimality. Afterwards, one of the local search moves RIS and RSS is chosen, the first one with probability jP and the second one with probability $1 - jP$. If the solution computed by the local search is better than the solution considered at the beginning of the current iteration, the next iteration continues the search from that solution. Otherwise, the output of the local search is considered as a starting point of the next iteration with probability $\exp(-(f(\Pi') - f(\Pi))/g(\mathcal{I}, \tau P))$, where $g(\mathcal{I}, \tau P) = \frac{\sum_{j \in J} \sum_{s \in S} p_{j,s}}{|J||S|10} \times \tau P$. The best solution found is updated whenever is necessary and it is provided as an output when the time limit is reached. The pseudocode of the procedure is given in Algorithm 1.

Algorithm 1 The IGT_NEH approach

```
Function IGT_NEH is
    Input: Time limit T_max, Parameters dS, τP, jP
    Output: Solution Π_best
    Π, γ ← NEH();
    Π^best ← Π;
    γ^best ← γ;
    while time limit T_max not reached do
        γ'' ← DEC_REC(γ, dS);
        r ← random value between 0 and 1;
        if r < jP then
            Π', γ' ← RIS(γ^best, γ'');
        else
            Π', γ' ← RSS(γ^best, γ'');
        if f(Π') < f(Π) then
            Π ← Π';
            γ ← γ';
            if f(Π') < f(Π^best) then
                Π^best ← Π';
                γ^best ← γ';
        else
            if r < exp −(f(Π') − f(Π))/g(I, τP) then
                Π ← Π';
                γ ← γ';
    return Π^best, γ^best;
```

4 Hybrid Approach

IGT_NEH implements a very different type of search compared to CP-based approaches. In fact, reference-based heuristics generate a new solution from scratch each time a different job permutation is considered, which occurs many times

in the different local search moves. This is the reason why these local search moves are not suitable for embedding into a CP system. Moreover, reference-based heuristics do not perform any local search that is devoted to optimizing the makespan by purely modifying how jobs are scheduled at the last stages. As an alternative, CP-based approaches branch on specific decisions, taken at each stage and consist of assigning a job to a machine such that the job starts on a certain time point.

The idea of the proposed hybrid approach is to exploit `IGT_NEH` to quickly find high quality solutions, while CP is used to intensify the search over the last stages of the schedule. The approach is based on two phases. The first phase consists of running `IGT_NEH` until the time limit $\frac{T_{max}}{2}$ is reached. Hence, we consider the CP model of the problem and we impose that the solution is identical to the one computed at the first phase up to stage ρ. The second phase consists of solving the resulting CP model, which basically re-optimizes the best solution found at the first phase, with respect to the stages that are successive to stage ρ. The model is solved using CP Optimizer, and its black-box search routine. This simplifies development compared to using the best performing solver in [1], which is SICStus Prolog, for which we would have to write a new specific search routine. The time limit of the second phase is $\frac{T_{max}}{2}$, such that the overall time limit of the approach is equal to T_{max}. Figure 1 depicts the structure of the proposed approach, which we denote as `HYBRID`.

5 Computational Experiments

This section describes the experiments conducted to assess the performance of `IGT_NEH` and `HYBRID`. The instances considered are the ones in [1], which refer to a realistic lane scenario in industry with 8 stages, where Stages 4 and 8 may be skipped by some jobs.

All experiments were performed on an Intel(R) Xeon(R) E5620 processor running at 2.40GHz with 32GB of memory. Please note that this machine is slightly slower than the machine used for the experiments in [1]. `IGT_NEH` was run by using the same parameter configuration as the one used in [15] ($dS = 2$, $\tau P = 0.5$, $jP = 0.4$). For the sake of performing a fair comparison, `HYBRID` is run in single-thread mode. Table 1 and Table 2 show the results obtained when the overall time limit is set to $T_{max} = 300$ seconds both for `IGT_NEH` and `HYBRID`. The values of each row of both tables are averages over 25 instances and 5 seeds. Please note that a dash indicates that no improved solutions were found within the time limit. Table 1 includes:

- the average objective value (obj value),
- the number of runs where an improvement was achieved with respect to `IGT_NEH` run for one half of the time limit (imp.),

for `IGT_NEH` run for 300 seconds, and `HYBRID` run for 300 seconds with $\rho \in \{0, 4, 6\}$. Only the average objective values are reported for `NEH`, and for `IGT_NEH` run for 150 seconds. Furthermore, the last two columns of both tables indicate

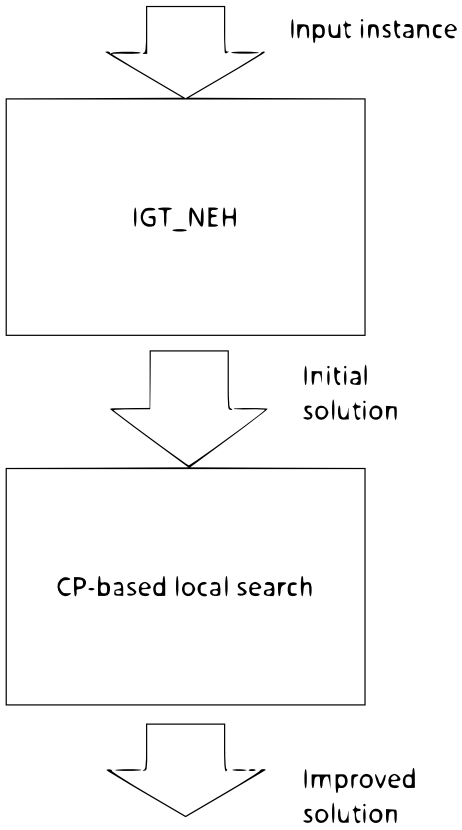


Fig. 1. Structure of the approach `HYBRID`.

the best results obtained with the different CP approaches published in [1], and the average of the lower bounds of the instances; see [1] for a description of the procedure used to compute these bounds.

Table 1. Average objective value and number of improved solutions per approach on different instance sizes, with the time limit $T_{max} = 300s$.

	NEH	IGT_NEH $\frac{T_{max}}{2}$	IGT_NEH T_{max}		HYBRID T_{max}, $\rho = 0$		HYBRID T_{max}, $\rho = 4$		HYBRID T_{max}, $\rho = 6$			
n	**obj**	**obj**	**obj**	**imp.**	**obj**	**imp.**	**obj**	**imp.**	**obj**	**imp.**	**CP21**	**LB**
20	66.72	62.78	62.76	3	62.75	**4**	62.77	2	62.776	1	**62.72**	61.88
25	68.84	64.32	64.26	7	64.26	**8**	**64.16**	6	64.30	3	**64.16**	62.84
30	72.52	66.81	66.68	16	66.74	8	**66.59**	**26**	66.78	4	66.68	64.12
40	78.2	72.37	72.26	14	72.18	21	**71.74**	**66**	72.12	40	72.56	65.32
50	85.42	78.94	78.81	21	78.31	20	**77.72**	**82**	78.12	43	78.4	67.24
100	115.56	109.78	109.51	29	-	-	**108.90**	**74**	109.26	54	113.04	94.72
200	176.24	171.69	170.08	73	-	-	**169.8**	**87**	170.98	60	176.72	153.08
300	239.12	238.54	**234.21**	93	-	-	235.24	**113**	240.96	75	240.96	214.96
400	298.82	298.82	298.54	32	-	-	**296.1**	**98**	298.07	61	303.16	275.36

Table 2 shows the average percentage (optimality) gap from the lower bound for each approach. It is interesting to note that, as with the previous CP results, the largest gap occurs for 50 or 100 jobs, while the optimality gap shrinks again for larger problem sizes.

The first key point that we can notice from Table 1 and Table 2 is that HYBRID is the best performing approach for most of the instance sizes. The best of its versions is the one with $\rho = 4$ stages for all the instances, but the ones that are very small ($n = 20$). Please note that in those instances the results published in [1] are slightly better, probably because of a more favorable computational setting (more powerful CPU, use of multiple threads). We can also note that IGT_NEH achieves a better result than HYBRID only for $n = 300$.

Table 3 provides an explanation for this, showing the average sum of squared deviation of the achieved objective values of runs performed with different seeds. This value is generally quite low, but very high for $n = 300$ and again very low for $n = 400$. For $n = 300$, the time limit stopped the search before the different runs converge to a common value, while for $n = 400$ there was not even enough time to find initial, but very variable, improvements in the local search routine over NEH.

Table 4 shows some experiments, performed with just one seed for an increased time limit, showing that HYBRID finds even better solutions given more time, and still outperforms IGT_NEH for the same time limit. Table 3 shows that we achieve a much lower average sum of squared deviations for $n = 300$, when the time limit is set to $T_{max} = 2400s$. Thus, an extension of the time limit helps improve the quality of the solutions provided by the proposed approaches and reduces the variance in the results. A closely related research question is to

Table 2. Average percentage gap from the lower bound per approach on different instance sizes, with the time limit $T_{max} = 300s$.

n	NEH	IGT_NEH $\frac{T_{max}}{2}$	IGT_NEH T_{max}	HYBRID T_{max}, $\rho = 0$	HYBRID T_{max}, $\rho = 4$	HYBRID T_{max}, $\rho = 6$	CP21
20	7.25%	1.44%	1.40%	1.40%	1.41%	1.43%	**1.34%**
25	8.72%	2.30%	2.22%	2.20%	**2.06%**	2.26%	2.06%
30	11.58%	4.02%	3.84%	3.93%	**3.71%**	3.98%	3.84%
40	16.47%	9.74%	9.60%	9.50%	**8.94%**	9.43%	9.98%
50	21.28%	14.82%	14.68%	14.14%	**13.48%**	13.93%	14.23%
100	18.03%	13.72%	13.51%	-	**13.02%**	13.31%	16.21%
200	13.14%	10.84%	10.00%	-	**9.85%**	10.47%	13.38%
300	10.10%	9.89%	**8.22%**	-	8.62%	10.79%	10.79%
400	7.85%	7.85%	7.76%	-	**7.00%**	7.62%	9.17%

Table 3. Average sum of squared deviation of the objective for IGT_NEH runs with different seeds on different instance sizes, with time limits $T_{max} = 300s$ and $T_{max} = 2400s$.

n	IGT_NEH ($T_{max} = 300$)	IGT_NEH ($T_{max} = 2400$)
20	0.064	-
25	0.272	-
30	0.336	-
40	0.464	-
50	0.544	-
100	0.88	-
200	8.624	0.624
300	38.56	3.616
400	0.688	5.588

establish what is the best way to split the time limit in HYBRID, which we will cover in our future studies.

Figure 2 shows how the approaches perform on the first instance with $n = 100$ when different time limits are used, with the exception of NEH which is executed once, until the procedure is completed. We notice that HYBRID outperforms IGT_NEH after 200 seconds and maintains its lead for larger time limits. Figure 3 shows that improvements for the different runs of HYBRID are less common after a time limit of 1000 seconds, which justifies the limit of 1200+1200 seconds for the experiments in Table 4.

In conclusion, the experiments are show a clear benefit in hybridizing IGT_NEH with a CP-based step, since HYBRID outperforms both approaches. This is probably due to the complementarity of the two approaches, which use different optimization strategies. In fact, the second phase of HYBRID, based on CP, intensifies the search on the last stages of the schedule. This aspect is not taken into account by reference-guided heuristics.

Table 4. Average objective values and gaps obtained with `IGT_NEH` and `HYBRID` on different instance sizes, with time limit $T_{max} = 2400s$.

	`IGT_NEH` (T_{max})		`HYBRID`(T_{max}, $\rho = 4$)		
n	**Obj value**	**Gap**	**Obj value**	**Gap**	**LB**
200	168.56	10.11%	**167.12**	**8.40%**	153.08
300	230.52	7.24%	**229.4**	**6.75%**	214.96
400	291.36	5.81%	**291.32**	**5.48%**	275.36

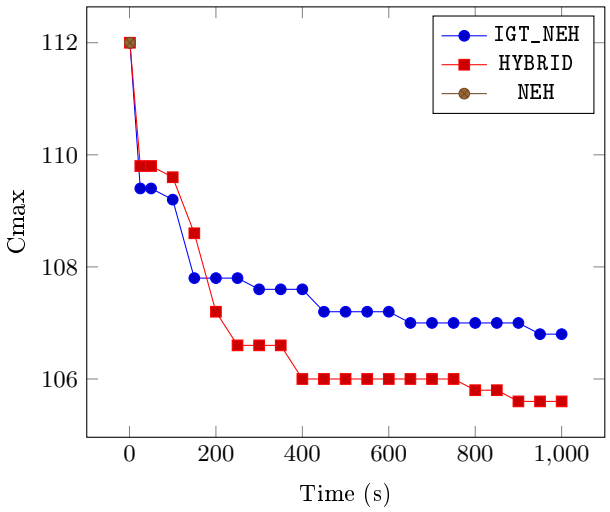


Fig. 2. Comparing average Cmax values for `IGT_NEH` and `HYBRID` over time for a single instance with 100 jobs

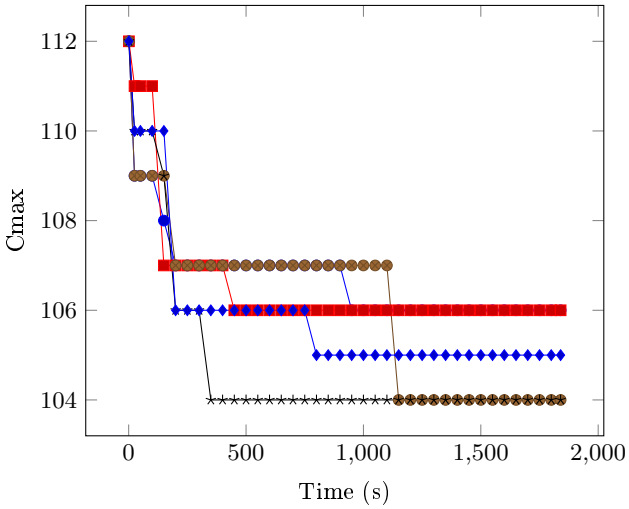


Fig. 3. Comparing Cmax value over time for five runs of `HYBRID` for a single instance with 100 Jobs

6 Conclusions and Future Work

This paper proposed a novel hybrid heuristic for the HFFTT problem, which combines a metaheuristic with constraint programming. Despite its simplicity, the proposed approach allowed us to improve the results obtained in [1] by means of different CP frameworks. At the same time, the hybrid also outperformed the state-of-the-art metaheuristic for the HFS, adapted to this problem variant.

An interesting research direction is to study how similar hybrid techniques perform on other variants of hybrid flowshop problems, including the HFP, by considering instances in reference datasets. The proposed hybrid approach can be seen as a CP-based decomposition heuristic, where the problem is decomposed with respect to the stages. Other alternative CP-based hybrid schemes may also be considered. One is to decompose the problem with respect to the jobs, fixing the first jobs for all stages to the value in the initial solution, and then rescheduling for the remaining jobs, even for the initial stages. Another alternative is to decompose the problem with respect to a certain time instant, by fixing all the tasks, regardless of stage, starting before, and solving the remaining problem. These and other decompositions will be explored in our future studies.

Acknowledgements

This publication has emanated from research conducted with the financial support of Science Foundation Ireland under Grant Number 16/RC/3918.

References

1. Armstrong, E., Garraffa, M., O'Sullivan, B., Simonis, H.: The Hybrid Flexible Flowshop with Transportation Times. In: Michel, L.D. (ed.) 27th International Conference on Principles and Practice of Constraint Programming (CP 2021). vol. 210, pp. 16:1–16:18. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany (2021). https://doi.org/10.4230/LIPIcs.CP.2021.16
2. Blum, C., Puchinger, J., Raidl, G.R., Roli, A.: Hybrid metaheuristics in combinatorial optimization: A survey. Applied Soft Computing **11**(6), 4135–4151 (2011). https://doi.org/https://doi.org/10.1016/j.asoc.2011.02.032
3. Della Croce, F., Garraffa, M., Salassa, F., Borean, C., Di Bella, G., Grasso, E.: Heuristic approaches for a domestic energy management system. Computers & Industrial Engineering **109**, 169 - 178 (2017). https://doi.org/https://doi.org/10.1016/j.cie.2017.05.003
4. Della Croce, F., Grosso, A., Salassa, F.: Minimizing total completion time in the twomachine noidle nowait flow shop problem. Journal of Heuristics (2019). https://doi.org/10.1007/s10732\-019\-09430\-z
5. Fanjul-Peyro, L., Perea, F., Ruiz, R.: Models and matheuristics for the unrelated parallel machine scheduling problem with additional resources. European Journal of Operational Research **260**(2), 482–493 (2017). https://doi.org/https://doi.org/10.1016/j.ejor.2017.01.002
6. Kergosien, Y., Mendoza, J.E., T'kindt, V.: Special issue on matheuristics. Journal of Heuristics **27**(1), 1–3 (Apr 2021). https://doi.org/10.1007/s10732-021-09472-2
7. Laborie, P., Rogerie, J., Shaw, P., Vilím, P.: IBM ILOG CP optimizer for scheduling. Constraints **23**(2), 210–250 (2018). https://doi.org/10.1007/s10601-018-9281-x
8. Lin, S.W., Ying, K.C.: Optimization of makespan for no-wait flowshop scheduling problems using efficient matheuristics. Omega **64** (2015). https://doi.org/10.1016/j.omega.2015.12.002
9. Maniezzo, V., Stützle, T., Voß, S. (eds.): Matheuristics - Hybridizing Metaheuristics and Mathematical Programming, Annals of Information Systems, vol. 10. Springer (2010). https://doi.org/10.1007/978-1-4419-1306-7
10. Naderi, B., Gohari, S., Yazdani, M.: Hybrid flexible flowshop problems: Models and solution methods. Applied Mathematical Modelling **38**(24), 5767–5780 (2014). https://doi.org/10.1016/j.apm.2014.04.012
11. Nawaz, M., Enscore, E.E., Ham, I.: A heuristic algorithm for the m-machine, n-job flow-shop sequencing problem. Omega **11**(1), 91–95 (1983). https://doi.org/https://doi.org/10.1016/0305-0483(83)90088-9
12. Rendl, A., Prandtstetter, M., Hiermann, G., Puchinger, J., Raidl, G.: Hybrid Heuristics for Multimodal Homecare Scheduling. In: Beldiceanu, N., Jussien, N., Pinson, É. (eds.) Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems. pp. 339–355. Springer Berlin Heidelberg, Berlin, Heidelberg (2012). https://doi.org/10.1007/978-3-642-29828-8_22
13. Ruiz, R., Vázquez Rodríguez, J.A.: The hybrid flow shop scheduling problem. European Journal of Operational Research **205**, 1–18 (2010). https://doi.org/10.1016/j.ejor.2009.09.024
14. Tang, T.Y., Beck, J.C.: CP and Hybrid Models for Two-Stage Batching and Scheduling. In: Hebrard, E., Musliu, N. (eds.) Integration of Constraint

Programming, Artificial Intelligence, and Operations Research. pp. 431–446. Springer International Publishing, Cham (2020). https://doi.org/10.1007/978-3-030-58942-4_28
15. Öztop, H., Fatih Tasgetiren, M., Eliiyi, D.T., Pan, Q.K.: Metaheuristic algorithms for the hybrid flowshop scheduling problem. Computers & Operations Research **111**, 177–196 (2019). https://doi.org/https://doi.org/10.1016/j.cor.2019.06.009