Expert Systems with Applications 39 (2012) 8149–8161

Contents lists available at SciVerse ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

New VNS heuristic for total flowtime flowshop scheduling problem

Wagner Emanoel Costa [a,*], Marco César Goldbarg [b], Elizabeth G. Goldbarg [b]

[a] *Programa de Pós-Graduação em Sistemas e Computação, UFRN/CCET/PPGSC – Campus, Universitário Lagoa Nova, Natal, RN, Brazil*
[b] *Departamento de Informática e Matemática Aplicada, UFRN/CCET/DIMAp – Campus, Universitário Lagoa Nova, Natal, RN, Brazil*

A R T I C L E I N F O

Keywords:
Flowshop
Scheduling
Total flowtime
Heuristics
VNS

A B S T R A C T

This paper presents a new Variable Neighborhood Search (VNS) approach to the permutational flowshop scheduling with total flowtime criterion, which produced 29 novel solutions for benchmark instances of the investigated problem. Although many hybrid approaches that use VNS do exist in the problems literature, no experimental study was made examining distinct VNS alternatives or their calibration. In this study six different ways to combine the two most used neighborhoods in the literature of the problem, named job interchange and job insert, are examined. Computational experiments were carried on instances of a known dataset and the results indicate that one of the six tested VNS methods, named *VNS*4, is quite effective. It was compared to a state-of-the-art evolutionary approach and statistical tests applied on the computational results indicate that *VNS*4 outperforms its competitor on most benchmark instances.

1. Introduction

In the permutational flowshop scheduling problem, there is a set of jobs $J = \{1,2,\ldots,n\}$. Each of n jobs has to be processed by a set of m machines $M = \{1,2,\ldots,m\}$, sequentially from the first machine to the last, in the same order. Each job j requires t_{jr} units of time on machine r. Each machine can process at most one job at any given time, and it cannot be interrupted. Each job is available at time zero and can be processed by at most one machine in any given time. Here the focus is to find the permutation of jobs $\Pi = \{\pi_1,\pi_2,\ldots,\pi_n\}$, such that the total completion time of jobs, named total flowtime (*TFT*), is minimized. Eq. (1) expresses mathematically the concept of total flowtime of a given permutation, Π, where $C(\pi_i,m)$ stands for the completion time of job in position i of Π, π_i. According to Rajendran and Ziegler (1997), *TFT* is an important objective in many real life manufacturing systems, for it minimizes holding costs.

$$TFT(\Pi) = \sum_{i=1}^{n} C(\pi_i, m) \tag{1}$$

The values of $C(\pi_i,m)$ can be evaluated using Eqs. (2)–(5). Eqs. (2) and (3) define the completion time relative to the first job in permutation Π, π_1. Eq. (2) defines the completion time of the first job, π_1, on the first machine as time required to complete its processing, $t_{1,1}$. It provides a base case for (3) which defines the completion time of π_1 on machine r, $1 < r \leqslant m$, as the completion time of π_1 on the previous machine, $C(\pi_1, r-1)$, plus the processing time of job π_1 on machine r, $t_{1,r}$.

$$C(\pi_1, 1) = t_{\pi_1,1} \tag{2}$$

$$C(\pi_1, r) = C(\pi_1, r-1) + t_{\pi_1,r} \quad \forall r \in \{2,\ldots,m\} \tag{3}$$

Eqs. (4) and (5) evaluate the completion times for each job π_i, $1 < i \leqslant n$. When $r = 1$, $C(\pi_i,1)$ is defined as the sum of the completion time of the previous job on the first machine, $C(\pi_{i-1},1)$, with the processing time of π_i, $t_{i,1}$. For all remaining machines, $1 < r \leqslant m$, completion time of job π_i, $C(\pi_i,r)$, $1 < i \leqslant n$, depends on two factors. First, the time on which job π_i concludes its processing on the previous machine, $r-1$, and therefore becomes available to be processed on machine r. Second, machine r can process job π_i only, if r has finished processing the previous job, π_{i-1}. If machine r has not concluded the previous job, π_{i-1}, then the processing of job π_i will start after machine r concludes job π_{i-1}. Eq. (5) expresses both factors and defines the completion time, $C(\pi_i,r)$, $1 < i \leqslant n$ and $1 < r \leqslant m$, as the sum of processing time $t_{i,r}$, with the greatest value between $C(\pi_i, r-1)$ and $C(\pi_{i-1}, r)$.

$$C(\pi_i, 1) = C(\pi_{i-1}, 1) + t_{\pi_i,1} \quad \forall i \in \{2,\ldots,n\} \tag{4}$$

$$C(\pi_i, r) = max\{C(\pi_i, r-1), C(\pi_{i-1}, r)\} + t_{\pi_i,r} \quad \forall i \in \{2,\ldots,n\} \quad \forall r \in \{2,\ldots,m\} \tag{5}$$

Due to the fact that the decision problem associated with *TFT* is NP-Complete in the strong sense when $m \geqslant 2$ (Graham, Lawler, Lenstra, & Kan, 1979), many heuristic approaches have been proposed to this problem such as constructive methods (Framinan, Leisten, & Ruiz-Usano, 2002; Liu & Reeves, 2001; Nagano & Moccellin, 2007;

* Corresponding author. Tel.: +55 84 3215 3814.
E-mail addresses: wemano@gmail.com (W.E. Costa), gold@dimap.ufrn.br (M.C. Goldbarg), beth@dimap.ufrn.br (E.G. Goldbarg).

Rajendran, 1993; Rajendran & Ziegler, 1997), local search methods (Dong, Huang, & Chen, 2009; Liu & Reeves, 2001) genetic algorithms (Nagano, Ruiz, & Lorena, 2008; Tseng & Lin, 2009; Xu, Xu, & Gu, 2011; Zhang, Li, & Wang, 2009a), ant colonies (Rajendran & Ziegler, 2004; Zhang, Li, Wang, & Zhu, 2009b), particle swarm optimization (Jarboui, Ibrahim, Siarry, & Rebai, 2008; Tasgetiren, Liang, Sevkli, & Gencyilmaz, 2007, 2007), bee colony optimization (Tasgetiren, Pan, Suganthan, & Chen, 2010), hybrid discrete differential evolutionary algorithm (Tasgetiren et al., 2010), VNS and EDA-VNS (Jarboui, Eddaly, & Siarry, 2009).

From the cited approaches, the works of Zhang et al. (2009a), Tseng and Lin (2009), Jarboui et al. (2009), Tasgetiren et al. (2010) and Xu et al. (2011) are the ones which produced the current state-of-art results.

A significant number among the state-of-art approaches combine two neighborhoods, named job insert and job interchange, in an internal VND procedure. The two neighborhoods are combined in the same order in the works of Zhang et al. (2009a), Tasgetiren et al. (2010) and Xu et al. (2011). Within flowshop's literature, the only work to test distinct ways to combine these neighborhoods is Pan, Tasgetiren, and Liang (2008), where a hybrid heuristic, particle swarm with VND, is devised for the no-wait flowshop problem. The experiments of Pan et al. (2008), test the hybridization of a PSO algorithm with two VND approaches, one starting with job insert and the other starting with job interchange. Both of them create all neighboring solutions before deciding to which solution to migrate. The results point out that the use of job interchange as the initial neighborhood provides the best performance on the tested instances of the no-wait flowshop. So far no study has been reported, for classical flowshop with total flowtime criterion, examining distinct combinations of neighborhoods.

The present study examines six distinct combinations of the two most common neighborhoods structures for the permutational flowshop scheduling using total flowtime criterion. Although, in the problem's literature, the use of VNS is common, the results of the experiments performed on instances of the suite created by Taillard (1993) suggested that, the most profitable combination of job interchange and job insert local search, is distinct from the combination of such local searches in literature. Considering that this new combination is a novel VNS for flowshop *TFT*, two questions arise: How does the new VNS perform when it is compared to state-of-the-art methods? Can current state-of-the-art methods benefit from using the new combination?

To address those questions, the novel VNS, labeled *VNS*4 is compared with two other approaches, one named AGA and the other named V4AGA. AGA is a genetic algorithm hybridized with VNS (Xu et al., 2011), which was claimed by its authors as superior algorithm when compared to other state-of-the-art methods. V4AGA is a novel hybrid approach proposed here, that replaces the VNS used in AGA by the one developed in this study. This experiment resulted in 34 novel solutions, 29 of then generated by *VNS*4.

The remaining of this paper is organized as follows. Section 2 reports a brief review of literature. Section 3 describes the VNS approaches as well as the describes the job insert and job interchange neighborhoods, the different ways to combine them and parameters tested in the experiments. Section 4 reports the experiments to determine which combination of neighborhoods performs best, using a subset of instances from Taillard (1993). Section 5 describes the computational experiments and results obtained by applying *VNS*4, AGA and V4AGA over all 120 instances of the data set. Section 6 presents some conclusions.

2. Literature review

This section reviews some methods proposed for PFSP with *TFT* criterion. Because the PFSP literature is vast this review is limited to only some constructive methods in literature, and metaheuristic approaches that constitute the current state-of-the-art of the problem.

The method of Rajendran and Ziegler (1997) evaluates the lower bound for each job available for assignment, and creates, m different solutions by changing as many machines as were considered in evaluating the lower bound. For instance, the first solution was constructed by sorting the jobs, in the non-decreasing order, by the weighted sum of the processing times of all m machines. The weights are defined in such a way that one unit of processing time of a machine, $r = j$, has greater impact than one unit of time of subsequent machines, $r > j$. Eq. (6) shows the formula for the unweighted total flowtime. The equation for the weighted total flowtime is slightly different, but the weighted case is not the topic of discussion in this work. The processing time of the first machine is deleted, and the jobs re-sorted considering only the processing times of the $m-1$ machines. This procedure is repeated, by deleting data from one machine at each iteration until the last solution is created, when only the processing time of the last machine is taken into account. The best solution among the m created ones is then submitted to a local search procedure using job insert neighborhood.

$$\sum_{r=j}^{m}(m-r+1)T_{ri} \quad j \in \{1,\ldots,m\} \tag{6}$$

The method $H(1)$ (Liu & Reeves, 2001) weights down two criteria: weighted sum of machine idle time (IT) and artificial flowtime (AT). The idle time criterion for selecting job i when k jobs were already selected (IT_{ik}), is defined by Eq. (7), where w_{rk} is calculated with Eq. (8). When the difference between $C(i,r-1)$ and $C(\pi_k,r)$ is positive, means machine r is idle. The weights, as defined in Eq. (8), stress that idle times on early machines are undesirable, for they delay the remaining jobs. Such stress is stronger if there are many unscheduled jobs (small value of k), but it becomes weak when the number k of scheduled jobs increases.

$$IT_{ik} = \sum_{r=2}^{m} w_{rk} max\{C(i,r-1) - C(\pi_k,r),0\} \tag{7}$$

$$w_{rk} = \frac{m}{r + k(m-r)/n - 2} \tag{8}$$

The artificial flowtime (AT_{ik}) of a candidate job i after k jobs were schedule refers to the *TFT* value obtained after including unscheduled job i, plus the time of an artificial job placed at the end of the job sequence. The processing time of the artificial job on each machine is equal to the average processing time of all unscheduled jobs, excluding job i, on the corresponding machine.

Both criteria, IT_{ik} and AT_{ik} are combined according to Eq. (9).

$$f_{ik} = (n-k-2)IT_{ik} + AT_{ik} \tag{9}$$

Also, Liu and Reeves (2001) propose a class of constructive heuristics named $H(x)$, where x is an integer, $1 \leqslant x \leqslant n$. Each variant differs in the number of solutions it produces which is given by x. While $H(1)$ produces only one solution, $H(2)$ produces two solutions and so on. These methods create new solutions by changing the initial job. Once the greedy criterion used in these heuristics is adaptive, different initial jobs produce different solutions. $H(1)$ uses the job with the best value according to the greedy criterion as the initial job. $H(2)$ creates two solutions using each of the two best evaluated jobs as the initial job. Thus, the first solution generated by $H(2)$ is exactly the same as the one generated by $H(1)$. The second solution uses the second best job as the initial job. A special case is $H(n/10)$ which uses the same principle, producing $n/10$ solutions with the best $n/10$ initial jobs. The work of Liu and Reeves (2001) also proposes the use of job interchange local search to further improve the greedy solution obtained.

Framinan et al. (2002) define a queue based on the sum of the processing times of each job. The job at the top of the queue is