Expert Systems with Applications 39 (2012) 13118–13126

Contents lists available at SciVerse ScienceDirect

Expert Systems with Applications

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

Hybridizing VNS and path-relinking on a particle swarm framework to minimize total flowtime

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

ARTICLE INFO

Keywords:
Flowshop
Scheduling
Total flowtime
Heuristics
PSO

ABSTRACT

This paper presents a new hybridization of VNS and path-relinking on a particle swarm framework for the Permutational Flowshop Scheduling Problem with total flowtime criterion. The operators of the proposed particle swarm are based on path-relinking and variable neighborhood search methods. The performance of the new approach was tested on the benchmark suit of Taillard, and five novel solutions for the benchmark suit are reported. The results were compared against results obtained using methods from literature. Statistical analysis favors the new particle swarm approach over the other methods tested.

1. Introduction

The Permutational Flowshop Scheduling Problem (PFSP) with the total flowtime criterion (*TFT*) is a NP-hard Combinatorial Optimization Problem (Garey, Johnson, & Sethi, 1976; Graham, Lawler, Lenstra, & Kan, 1979) that deals with the scheduling of a set of jobs, *J*, through a set of machines, *M*. Every job has to be processed on all machines following the same machine sequence. Each job requires a definite machine-processing time and can be processed on one machine at a time. Each machine processes only one job at a time. Once a machine starts processing a job, no preemption is allowed, and the machine becomes available as soon as the operation is finished. The goal is to produce a schedule of jobs such that the sum of completion times is minimized. Many methods have been proposed to address this problem. There are constructive methods (Framinan, Leisten, & Ruiz-Usano, 2002; Framinan & Leisten, 2003; Laha & Sarin, 2009; Liu & Reeves, 2001; Nagano & Moccellin, 2007; Rajendran & Ziegler, 1997), local search methods (Dong, Huang, & Chen, 2009), iterated greedy approach (Pan, Tasgetiren, & Liang, 2008), genetic algorithms (Duan, Zhang, Qiao, & qing Li, 2011; Xu, Xu, & Gu, 2011; Zhang, Li, & Wang, 2009a), ant colonies (Rajendran & Ziegler, 2004; Zhang, Li, Wang, & Zhu, 2009b), particle swarm optimization (Liao, Tseng, & Luarn, 2007; Jarboui, Ibrahim, Siarry, & Rebai, 2008; Tasgetiren, Liang, Sevkli, & Gencyilmaz, 2007), bee colony optimization (Tasgetiren, Pan, Suganthan, & Chen, 2010), VNS (Costa, Goldbarg, & Goldbarg, 2012; Jarboui, Eddaly, & Siarry, 2009), hybrid EDA approaches (Jarboui et al., 2009; Zhang & Li, 2011), hybrid discrete differential evolutionary algorithm (Tasgetiren et al., 2010) and parallel simulated annealing (Czapinski, 2010).

The particle swarm optimization (PSO) is a metaheuristic proposed by Eberhart and Kennedy (1995) for continuous optimization. Due to its simplicity it has been adapted to many discrete optimization problems including the PFSP (e.g. Liao et al., 2007; Jarboui et al., 2008; Tasgetiren et al., 2007).

The PSO approach was based on models explaining the synchronous movements in a flock of birds. In those models the birds attempt to keep an optimal distance from each other so they can be close enough to profit over the discoveries and previous experience of a neighboring bird, and at the same time avoiding the competition for food (Eberhart & Kennedy, 1995).

The proposed discrete PSO is based on the work of Goldbarg, Goldbarg, and de Souza (2008) for the Traveling Salesman Problem. Therefore, the cited metaphor is translated into an optimization method as follow. Agents, named particles, fly over the solution space. A particle occupies a position on the solution space. The position of a particle represents a valid solution currently under exam, it is encoded as a permutation of jobs (Π), and it has a objective value associated to it, named $TFT(\Pi)$. Besides knowing their current position, the particle knows the best site (solution) it previously visited, knows the current position of a neighboring particle, and the best sites previously visited by a neighbor. On any given iteration of the method, each particle will compromise in doing one of the following actions: (*a*) to explore the solution space on its own; (*b*) to move towards the current site of a neighboring bird; (*c*) to move towards the best site previously visited by a neighboring bird. A probability is assigned to each possible action. During execution those probabilities are updated, the update process takes into account the quality of the last solution obtained through each action.

The actions of the particles are implemented using search operators such as variable neighborhood search procedure (VNS) and path-relinking. Given a particle *A*, if *A* chooses to explore the search space on its own, *A* copies the configuration of the best previous site

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

0957-4174/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.eswa.2012.05.090

it visited $\Pi_{A,Best}$ to its current position $\Pi_{A,Curr}$, a VNS procedure is executed over Π', the resulting solution becomes the current position of A, $\Pi_{A,Curr}$. If the particle chooses to move towards a neighboring solution B, the second possible action, a combination of path-relinking procedure with VNS is executed. Initially, the path-relinking gradually transforms the current position $\Pi_{A,Curr}$ (solution) of particle A in the position $\Pi_{B,Curr}$, occupied by the neighboring particle B. If during this process an intermediate solution $\Pi'_{A,Curr}$ is found, such that the value of its objective function is smaller than the objective function values of $\Pi_{A,Curr}$ and $\Pi_{B,Curr}$, then path-relinking is interrupted and the VNS procedure is executed over $\Pi'_{A,Curr}$, resulting in $\Pi''_{A,Curr}$. After VNS finishes, path-relinking is resumed transforming $\Pi''_{A,Curr}$ into $\Pi_{B,Curr}$. A new interruption may occur if a solution better than both $\Pi''_{A,Curr}$ and $\Pi_{B,Curr}$ is found. The action concludes when no further improvement is found during path-relinking. The third action is implemented similarly to the second action, however in this case the target position is the best site visited by the neighboring particle B ($\Pi_{B,Best}$) instead of its current position ($\Pi_{B,Curr}$).

The remainder of this article is organized into three sections. In Section 2 the description of the problem is given. Section 3 describes the hybrid algorithm proposed for the PFSP with *TFT* criterion. Section 4 addresses the experiments carried out to tune parameters involved in proposed approach. Section 5 presents computational experiments comparing the proposed PSO with two state-of-the-art methods from literature. Statistical analysis over the results indicates significant differences favoring of the proposed PSO over two state-of-the-art approaches. Finally, some conclusions are presented in Section 6.

2. Problem

As stated earlier, in the permutation flowshop context a set of jobs $J = \{1,\ldots,n\}$ will be processed by a set of machines $M = \{1,\ldots,m\}$ in sequence. Job j has a processing time of T_{jr} on machine r, $1 \leqslant j \leqslant n$, $1 \leqslant r \leqslant m$. Let the permutation $\Pi = \{\pi_1, \pi_2, \ldots, \pi_n\}$ denotes the job-processing order, and π_i corresponds to the ith job on the schedule Π. The completion time of job π_i on machine r, denoted by $C(\pi_i, r)$, is given by the time elapsed since the first job begins to be operated on the first machine until job π_i is completed on machine r. $C(\pi_i, r)$ is proper evaluated through Eqs. (1)–(4), where Eq. (1) refers to the conclusion time of the first job scheduled on the first machine, and Eq. (2) refers to the conclusion time of the first job scheduled on the remainder machines. Analogously, Eq. (3) evaluates the completion time of job π_i, $1 < i \leqslant n$, on the first machine ($r = 1$), whereas Eq. (4) evaluates the completion time of job π_i, $1 < i \leqslant n$, on the remainder machines, $1 < r \leqslant m$. Based on Eqs. (1)–(4), the total flowtime value of a given permutation Π ($TFT(\Pi)$) is defined as the sum of completion times on the last machine (Eq. (5)).

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

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

$$C(\pi_i, 1) = C(\pi_{i-1}, 1) + T_{\pi_i,1} \quad \forall i \in \{2,\ldots,n\} \tag{3}$$

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

$$TFT(\Pi) = \sum_{i=1}^{n} C(\pi_i, m) \tag{5}$$

3. Discrete particle swarm optimization

This section describes the proposed discrete PSO to optimize the total flowtime, and there are five subsections. The first subsection shows the pseudo-code of the proposed PSO and discusses four procedures to be defined. Each subsequent subsection explains in details each procedure and their tuning parameters.

3.1. Pseudo-code

Algorithm 1 exhibits the pseudo-code of the proposed method. The PSO procedure depends on other procedures, at first the pseudo-code of PSO is explained and the subsequent paragraphs detail the procedures on which PSO depends on. The first line of the algorithm initializes the set of particles (*Particles*), meaning that the position of each particle is set to an individual solution. The best visited site of a given particle is also initialized with its current position. The main body of PSO lies on the loop from lines 2–20. Usually the main loop on PSO is repeated until the stop criterion is satisfied, in this case the stop criterion was defined as a time limit of $0.4 \times n \times m$ s, as the fastest heuristics for minimizing *TFT* in PFSP context use the same stop criterion (e.g. Jarboui et al., 2009; Zhang & Li, 2011; Xu et al., 2011; Costa et al., 2012). In line 4, probabilities v_a, v_b and v_c are defined in the procedure *ComputeProbabilities* standing for the probabilities of particle P choses the first, second or third action, respectively. In line 5 a random number is picked from the interval [0,1] and stored in variable *Action*. If the value of *Action* is smaller or equal to v_a (line 6), then particle P explores the search space in function *ExploreAlone*, otherwise P will move towards a neighbor or towards the best site of a neighbor. In both cases a destiny must be selected, this is done in lines 9–12. In line 9, the neighbor, of which either its current position or its best site will be used as a guide, is selected randomly. The loop from line 10 to line 12 ensures that a neighbor is selected. If the value of *Action* is greater than v_a and smaller or equal to v_b then particle P will move from its current position ($\Pi_{P,Curr}$) towards a site currently occupied by a neighbor ($\Pi_{Target,Curr}$). That move is made in procedure *MoveToNeighbor*, line 14. If the value of *Action* is greater than v_b then P moves towards the best site a neighboring particle has occupied, line 16. The main loop finishes when the time limit is reached. In lines 21–26 the algorithm finds the best solution ever achieved by a particle and returns it in *BestSol*, line 27.

Algorithm 1. Discrete PSO

```
1: Initialize the set of particles Particles
2: repeat
3:    for each particle P with position Π_{P,Curr} and best visited
      site Π_{P,Best} do
4:       (v_a, v_b, v_c) ← ComputeProbabilities()
5:       Action ← random number ∈ [0,1]
6:       if Action ⩽ v_a then
7:          ExploreAlone(P)
8:       else
9:          Target ← random number ∈ {1,2,...,|Particles|}
10:         while Target = P do
11:            Target ← random number ∈ {1,2,...,|Particles|}
12:         end while
13:         if v_a < Action ⩽ v_b then
14:            MoveTowards(Π_P, Π_{Target,Curr})
15:         else
16:            MoveTowards(Π_P, Π_{Target,Best})
17:         end if
18:      end if
19:   end for
20: until time limit of 0.4 × n × m s is reached
21: BestSol ← Π_{1,Best}
22: for P = 2 to |Particles|
23:    if TFT(Π_{P,Best}) < TFT(BestSol) then
24:       BestSol ← Π_{P,Best}
25:    end if
26: end for
27: return BestSol
```