Submitted to *Operations Research*
manuscript (Please, provide the manuscript number!)

Balancing the average weighted completion times in a scheduling problem with two classes of jobs: a genetic algorithm-based heuristic.

Matteo Avolio
Department of Mathematics and Computer Science, University of Calabria, Rende (CS), Italy, matteo.avolio@unical.it

Balancing the average weighted completion times among two classes of jobs (BAWCT) could be interpreted as a cooperative multiagent scheduling problem. BAWCT was introduced in [1] and explored in [2] where the authors proved its NP-hardness, providing a Lagrangian heuristic algorithm for solving it. Since their approach requires to solve, at each iteration, a Linear Programming problem that it's not a very difficult task but however requires some computational time, in this work we propose a genetic algorithm-based heuristic to speed up the resolution process.
Numerical results are presented on the same datasets of [2] to empirically show the efficiency of the proposed approach.

Key words: Scheduling; single-machine; multiagent; genetic algorithm
History:

1. Introduction

Allocating resources to tasks over given time periods with the aim of optimizing one or more objectives is what the theory of scheduling focus on [Pinedo]. Scheduling, as a decision making process, plays a very important role in a lot of contexts such as manufacturing, production, transportation and distribution. Unfortunately solving this kind of problems often is not easy but, however, it depends on the assumptions made on the problem.
In the first years of the century, with publication of Agnetis et al (2004) and Baker and Smith (2003), a new scheduling model was introduced to take into account the presence of two or more agents. In multiagent scheduling problems, each agent has its own set of tasks to be processed and its own objective function to be optimized. The difficulty of this kind of problems arises from the fact that the agents share the processing resources.
Baker and Smith (2003) face up the problem by using three different evaluation criteria: minimizing makespan, minimizing maximum lateness, and minimizing total weighted completion time. From a complexity view point, they proved the problem to be polynomial solvable according to any one of the three mentioned criteria but, additionally, they shown that when minimizing a mix of them, the problem becomes NP-hard. The multiagent assumptions find also application in a lot of practical contexts. For example, Peha (1995) address the problem of minimizing the weighted number of tardy jobs in a real-time systems and integrated-services networks while Arbib (2004,A competitive scheduling problem and its relevance)), in a telecommunication system scenario with two users, focus on the maximization of on-time data packets transmitted to one user, while guaranteeing a certain amount of on-time data packets to the other
The majority of multiagent scheduling problem are of competitive type since agents purely compete with each other to allocate resources to their tasks that only contribute to their own objective function.
In this work, instead, we tackle a scheduling problem whose could be interpreted as a two agents problem of cooperative type since each job contributes to the same objective function aimed at balancing the average weighted completion times of the two agents.
This problem finds application in a lot of scenarios characterized by a decision maker who,

for some reasons (e.g. economic, of efficiency, of reliability), wishes to balance the average completion times of two classes of jobs. Let's consider, for example, a transportation firm that uses a drone for delivering packages and has to stock two different companies. In this scenario, the processing times represent the shipping times whose depend on the depot-company distance and on the weight of the corresponding cargo, the latter acting on the speed of the drone, while the weight associated with each job could be, for example, the urgency specified by a company for that package. From the point of view of the transportation firm, it's reasonable to schedule deliveries in order to balance the average weighted time of the two companies.
This problem was introduced in [1] where the authors considered its basic version by assuming all the jobs having the same processing time and unitary weight. In [2] the problem was generalized by considering all the jobs having different processing times and weights. Unfortunately, for the generalized version of the problem, the authors proved its NP-hardness, providing a Lagrangian relaxation based algorithm to solve it heuristically. Since their approach requires, at each iteration, to solve a Linear Programming problem that it's not too hard but it requires some computational time, in this work we propose a genetic algorithm based heuristic to speed up the resolution process.
This work is organized as follows. In the next section we formally state the problem, reporting a nonsmooth formulation as a variant of the quadratic assignment problem, and we provide the state-of-the-art results for it. In Section 3 we propose a genetic algorithm based approach to heuristically solve the problem, describing in detail all the phases of the considered approach. In Section 4 we present some numerical results obtained on the same datasets considered in literature to empirically show the efficiency of the proposed algorithm and we conclude with Section 6 in which some conclusions are drawn.

2. Problem Definition and Literature Review

Let A and B be two different classes of jobs with n_A and n_B being their cardinalities. Let $J_A = \{1,\dots,n_A\}$ and $J_B = \{n_A+1,\dots,n_A+n_B\}$ be the indices sets of A and B, respectively. For each job $j \in J_A \cup J_B$, let p_j be its processing time and w_j its weight.
The problem of balancing the average weighted completion times (BAWCT)[2] of class A and B can be stated as follow:

$$\min \left| \frac{\sum_{j\in J_A} C_j w_j}{n_A} - \frac{\sum_{j\in J_B} C_j w_j}{n_B} \right|, \qquad (1)$$

where C_j represents the completion time of job j. From a mathematical programming point of view, in order to formulate the BAWCT as an optimization problem, we proceed as follows. We define the following decision variables:

$$x_{jt} \triangleq \begin{cases} 1 & \text{if job } j \text{ is assigned to position } t \\ 0 & \text{otherwise,} \end{cases}$$

for $j \in J_A \cup J_B$ and $t = 1\dots, n_A + n_B$.

Taking into account that the completion time of a job j scheduled in position t is

$$p_j + \sum_{l\in J_A\cup J_B} \sum_{k=1}^{t-1} p_l x_{lk},$$

problem (1) can be formulated as follows:

$$\begin{cases} \min\limits_x \left| \dfrac{1}{n_A} \left(\displaystyle\sum_{j\in J_A} \sum_{t=1}^{n} w_j \left[p_j + \sum_{l\in J} \sum_{k=1}^{t-1} p_l x_{lk} \right] x_{jt} \right) \right. \\ \left. - \dfrac{1}{n_B} \left(\displaystyle\sum_{j\in J_B} \sum_{t=1}^{n} w_j \left[p_j + \sum_{l\in J} \sum_{k=1}^{t-1} p_l x_{lk} \right] x_{jt} \right) \right| \\ \displaystyle\sum_{t=1}^{n} x_{jt} = 1 \quad j \in J \\ \displaystyle\sum_{j\in J} x_{jt} = 1 \quad t = 1\dots n \\ x_{jt} \in \{0,1\} \quad j \in J, \quad t = 1\dots n, \end{cases} \qquad (2)$$

where the constraints are the classical assignment constraints, which impose a bijection between jobs and positions. Problem (2) is a nonsmooth integer optimization problem and represents a sort of quadratic assignment problem due to the quadratic terms in the objective function. In [2] the authors, after having proved its NP-hardness, provided a linearization of (2)

based on the well known Glover Linearization and then they heuristically solved the linearized version using a Lagrangian relaxation based approach. The limit of their technique, even if the obtained results are valuable, consists on the fact that, at each iteration, a Linear Programming problem has to be solved. It's well known that it can be solved in polynomial time but, when the size of the problem increases (i.e. n_A and n_B become very large), its resolution has a not negligible impact on the overall computational time. For this reason, in the next section, we propose a genetic algorithm-based heuristic that, from the empirical results, has proved to be able to speed up the resolution process.
It's worth noting that, in [1], the authors tackled a simplified version of BAWCT in which they assumed $w_j = 1, j = 1, \ldots, n_A + n_B$ and $p_j = p, j = 1, \ldots, n_A + n_B$. They proved that, by mean of these assumptions, the problem reduces to a particular instance of the well known subset sum and it becomes solvable in linear time, or constant time if the job-position assignment is not explicitly considered.

3. A Genetic Algorithm Heuristic

Genetic Algorithms (GAs)[book] are heuristic search approaches successfully applied to many NP-hard optimization problems. They follow the evolution paradigm, i.e. starting from an initial population, they apply genetic operators in order to produce offsprings, trying to make them more fit than their ancestors, hopefully increasing the overall fitness of the population from one generation to another.
In GAs setting, to each individual corresponds a genotype representing a possible solution to the optimization problem and a fitness value representing its "goodness"; then, from one generation to another, GAs wish to generate new individuals with higher fitness values.
The basic schema of GAs is reported in Algorithm 1.
Even if in literature there are plenty of suggestions about how to implement all the phases of Algorithm 1, in general each of them can be adapted and tailored to the problem dealing with; this flexibility makes GAs attractive for many optimization problems in practice.
Before to describe in detail how we implemented each phase, its worth to specify some additional steps we performed. In particular, after mutation, we applied a local search to improve the fitness value. Additionally, at each generation, we completely replace the old population with the new one, just mantaining the fittest current individual in order to have a monotonic behaviour of the overall fitness. These steps are used quite always in GAs in order to have a faster convergence to optimal solutions.
Finally, as exit conditions we used a time limit of 1800 seconds like in [2]. Of course, we stop the algorithm also if a null upper bound is found since, by the definition of (1), it trivially corresponds to an optimal solution.

Algorithm 1: Genetic Algorithm

```
initialize population
while not end conditions do
    while new population uncomplete do
        selection
        crossover
        mutation
    end
    population replacement
end
```

3.1. Encoding and Fitness Evaluation

The first step in the implementation of a GA is the choice of a suitable encoding for solutions. Since we are dealing with a scheduling problem, the standard way to represent a solution is a vector of values.
Given a solution S, we encode it in a vector $V(S)$ of size $n_A + n_B$ such that $V[t] = j$ if and only if job j is scheduled in position t in the solution S.
Example. Let's assume $n_A = 1$ and $n_B = 3$.
The following vector of size $n_A + n_B = 4$

3	1	4	2

encodes the solution in which job 1, belonging to class A, is scheduled in second position, while jobs 2, 3 and 4, belonging to class B, are scheduled in fourth, first, and third position, respectively.
Concerning with fitness evaluation, since individual with higher fitness are preferred but we are dealing with a minimization problem, we proceed as follows.
Let S be a solution for BAWCT. We compute

$f(S)$, i.e. its corresponding objective function value, by using (1) and then we set $fitness_S = \frac{1}{f(S)}$.
In this way, when $f(S) \to 0$ (that is a trivial lower bound for (1)) follows that $fitness_S \to \infty$.

3.2. Initial population

Once the encoding strategy has been defined, the next step to be performed regards the generation of the initial population.
In our approach, we tried three different techniques:

- *Random generation.* Let $N = \{1, \ldots, n_A + n_B\}$. The genotype of each individual is given by a random permutation of N that trivially represents a feasible solution for BAWCT.
- *Alternated generation.* In order to avoid inserting in the initial population completely unbalanced solutions, the genotype of each individual is given by alternating, until one of the two sets is completely scheduled, the jobs of class A and B. Afterwards, potentially remaining jobs are accomodated at the end.
- *Bidirectional generation.* A coppie uno all'inizio e uno alla fine in base alla somma corrente

3.3. Selection

Selection is a crucial step in designing a GA since it defines the chance of a given individual to participate in the reproduction process. Since a convergence to optimal solutions is desired, it's recommended to give an higher chance to individuals with an higher fitness value.
In this work, we used three different well-known selection techniques:

- *Roulette wheel.* With this technique, the selection is performed by simulating a roulette wheel in which each individual is represented by a portion of the roulette whose size depends on its fitness value. In this way, each individual I has a probability p_I to be selected that is proportional to its fitness.
In particular, we set

$$p_I = \frac{fitness_I}{\sum_i fitness_i}.$$

- *Binary tournment.* From two randomly selected individuals, the one having highest value of fitness is chosen for reproduction.
- *k-tournment.* It's the same as the previous with the only difference that k individuals are involved in the tournment.

3.4. Crossover

Crossover operators, just like mutation ones, are used to produce new solutions in the chosen representation and allow the explore the solution space.

3.5. Mutation

3.6. Local Search

4. Numerical Results

Scelta dei parametri, commento sull'utilità dell'inizializzazione intelligente con relativa tabella della fitness iniziale.

n	n_A	n_B	Exit(s)	Iter
20	10	10	0,028	19,1
30		20	0,017	6,4
40		30	0,028	5,6
40	20	20	0,028	5,7
50		30	0,08	9,6
60		40	0,068	3,7
60	30	30	0,067	3,6
70		40	0,19	7
80		50	0,32	7,7
80	40	40	0,091	2,2
90		50	0,748	11,6
100		60	0,347	4,4

Table 1 Small test problems, $p \in [1, 25]$.

n	n_A	n_B	Exit (s)	Iter
100	50	50	0,416	5,5
150		100	0,863	3,6
200	100	100	1,705	2,8
250		150	6,121	4,4

Table 2 Medium test problems, $p \in [1, 50]$.

n	n_A	n_B	Exit (s)	Iter
300	150	150	14,566	5,3
350		200	62,988	11,7
400	200	200	24,737	3,2
450		250	135,138	11,3
500	250	250	49,91	2,8

Table 3 Large test problems, $p \in [1, 100]$.

$p \in [1, 3n]$	n	n_A	n_B	Time for Incumbent (s)	Iter	Absolute Error (%)	Solved to Opt
$p \in [1, 180]$	60	30	30	1,83	119,8	0	20/20
$p \in [1, 300]$	100	50	50	6,207	81,9	0	20/20
$p \in [1, 600]$	200	100	100	202,428	322,5	0	20/20
$p \in [1, 900]$	300	150	150	782,097	339,4	0,002	16/20
$p \in [1, 1200]$	400	200	200	1087,752	183	0,004	11/20
$p \in [1, 1500]$	500	250	250	960,657	91,5	0,008	2/20

Table 4 Test problems with large ranges

5. Conclusions

References