Simulated Annealing as an Intensification Component in Hybrid Population-Based Metaheuristics

Davide Anghinolfi and Massimo Paolucci
Department of Communication, Computer and Systems Sciences
University of Genova
Italy

1. Introduction

The use of hybrid metaheuristics applied to combinatorial optimization problems received a continuously increasing attention in the literature. Metaheuristic algorithms differ from most of the classical optimization techniques since they aim at defining effective general purpose methods to explore the solution space, avoiding to tailor them on the specific problem at hand. Often metaheuristics are referred to as "black-box" algorithms as they use limited knowledge about the specific problem to be tackled, instead usually taking inspiration from concepts and behaviours far from the optimization field. This is exactly the case of metaheuristics like simulated annealing (SA), genetic algorithm (GA), ant colony optimization (ACO) or particle swarm optimization (PSO). Metaheuristics are based on a subset of features (e.g., the use of exploration history as short or long term memory, that of learning mechanisms or of candidate solution generation techniques) that represent a general algorithm fingerprint which usually can be easily adapted to face different complex real world problems. The effectiveness of any metaheuristic applied to a specific combinatorial problem may depend on a number of factors: most of the time no single dominating algorithm can be identified but several distinct mechanisms exploited by different metaheuristics appear to be profitable for searching high quality solutions. For this reason a growing number of metaheuristic approaches to combinatorial problems try to put together several techniques and concepts from different methods in order to design new and highly effective algorithms. Hybrid approaches in fact usually seem both to combine complementary strengths and to overcome the drawbacks of single methods by embedding in them one or more steps based on different techniques. As an example, in (Anghinolfi & Paolucci, 2007a) the SA probabilistic candidate solution acceptance rule is coupled with the tabu list and neighbourhood change mechanisms respectively characterizing tabu search (TS) and variable neighbourhood search (VNS) approaches to face parallel machine total tardiness scheduling problems. Several surveys exist proposing both classifications of metaheuristics and unified views of hybrid metaheuristics (e.g., (Blum & Roli, 2003), (Doerner et al., 2007), (Raidl, 2006) and (Talbi, 2002)). We would avoid to replicate here the various definitions and classifications through which the different approaches can be analysed and organized (the interested reader can for example refer to (Blum & Roli, 2003)

Source: Simulated Annealing, Book edited by: Cher Ming Tan, ISBN 978-953-7619-07-7, pp. 420, February 2008, I-Tech Education and Publishing, Vienna, Austria

for a valuable review). However, we should underline few basic concepts that allow us to focus on the different characteristics of the kinds of methods used in the hybrid algorithms presented in this chapter. SA, ACO and PSO are all stochastic algorithms, but SA is commonly classified as a *trajectory-based* method since it determines at each iteration a new single current solution, whereas ACO and PSO are *population-based* methods since they explore at each iteration a set of distinct solutions which they make evolve iteration after iteration. The concept behind these two *population-based* methods is that the overall exploration process can be improved by learning from the single exploring experiences of a population of very simple agents (the ants or the particles). As will be cleared in the following of the chapter, ACO explicitly exploits a learning mechanism in order to identify, iteration after iteration, which features should characterize good, i.e., the most promising, solutions. If in ACO the communication among the exploring agents (the ants) is indirect, PSO, on the other hand, drives the search of the population of agents (the swarm of particles) on the basis of simple pieces of information (e.g., where the current best is located), making the agents moving towards promising solutions. Therefore, both ACO and PSO use memory structures, more complex in ACO, simpler in PSO, to elaborate their exploration strategies; agents in ACO and PSO perform a learning or information driven sampling of the solution space that could in general be considered wide but also quite coarse, and that can be trapped in local optima (the so-called *stagnation* (Dorigo & Stutzle, 2004)). SA, on the other hand, is a memoryless method which combines the local search aptitude of exploring in depth regions in the solution space with the ability, ruled by the *cooling schedule* mechanism, of escaping from local optima. From this brief overview the possible advantage of coupling the different complementary abilities of the two types of metaheuristics should begin to emerge. Therefore in this chapter our purpose is to focus the attention on hybrid population-based metaheuristic algorithms with a specific reference to the use of SA as a hybridizing component. Then, according to the classification proposed in (Raidl, 2006), the kind of hybrid algorithms here considered result from the combination of two distinct metaheuristics (the "what is hybridized" aspect) among which a low-level strong coupling is established (the "level of hybridization" aspect), in particular the execution of SA is interleaved with the iterations of the population-based metaheuristics (the "order of execution" aspect) so that SA can be viewed as an integrated component of these latter (the "control strategy" aspect).
Several works recently appeared in the literature show the interest of embedding SA into population-based approaches as ACO, PSO and GA. Examples of PSO hybridized by incorporating SA intensification can be found in (Liu et al., 2008), where the proposed hybrid PSO (HPSO), which includes a probabilistically applied local search (LS) and a learning-guided multi-neighbourhood SA, is applied to makespan minimization in a permutation flow shop scheduling problem with the limited buffers between consecutive machines; in (He & Wang, 2007), where constrained optimization problems are faced by a HPSO which applies the SA search from the best solution found by the swarm in order to avoid the premature convergence; in (Li et al., 2006), where the hybrid algorithm, named PSOSA, is used for non-linear systems parameter estimation; in (Ge et al., 2007) where the HPSO is used to face the job shop scheduling. Differently, in (Xia & Wu, 2005) multi-objective flexible job shop scheduling problems are confronted by a hierarchical approach exploiting PSO to assign operations to machines and then SA to schedule operations on each machine. Hybrid ACO approaches, which combine pheromone trail based learning

mechanism with the SA search ability of escaping from local optima, are proposed for example in (Demirel & Toksarı, 2006) for the quadratic assignment problem and in (Yuanjing & Zuren, 2004) for flow-shop scheduling problems. Finally, in (Yogeswaran et al., 2007) a hybrid metaheuristic named GASA, which combines GA and SA, is used to solve a bi-criterion machine loading problem in flexible manufacturing system.

In this chapter we would highlight the effectiveness of embedding a trajectory method, i.e., SA, as intensification method of population-based algorithms, i.e., ACO and PSO. Many works in the literature witnessed the fundamental role for population-based approaches, as ACO, PSO or GA, of an intensification phase which usually corresponds to a local search (LS) exploration (Blum & Roli, 2003). However, a well-known and common characteristic of trajectory methods, as SA, VNS or TS, is their ability of overcoming the LS limitation of being trapped in local optima. For this reason the role of this class of powerful methods goes beyond that of a local intensification procedure, since they allow the calling population-based method to be "re-directed" towards portions of the solution space which may not be confined to the basin of attraction of a local optimizer. Then, we can view the hybrid algorithms discussed in this chapter as composed by a main population-based component which exploits a second level subordinate SA procedure in order to deeply explore (intensify) the neighbourhood of one (or more) promising solution, as well as escaping from such a neighbourhood when it includes a local optima attractor (diversify). On a symmetric standpoint, we could also consider these hybrid metaheuristics as an *iterated* trajectory method, i.e., an iterated SA, whose (promising) starting solutions are determined at the beginning of each iteration by a population-based algorithm. This latter algorithm in fact, exploiting memory and/or learning mechanisms, performs a sort of solution perturbation or shaking, possibly driving the SA search to focus on alternative promising regions of the solution space. In this case we can consider the population-based algorithm as an effective memory and learning based diversification device for SA. Whatever standpoint one would prefer, we believe that the effectiveness of the overall resulting algorithm emerges from the interaction of the complementary capabilities of the methods of the two different classes, that is, according to (He & Wang, 2007), from the balance of the intensification and diversification components included in them. An important aspect to be taken into account when designing the interaction mechanism between the population-based and the trajectory (i.e., SA) components of the hybrid algorithm regards how to identify the solutions which are worth to intensify; therefore in this chapter, we will also discuss several alternative strategies available to this end, pointing out their possible different effectiveness and computational burden.

The rest of this chapter is organized as follows. First in the Section 2 we briefly present the two scheduling problems used as reference to analyse the behaviour of the hybrid metaheuristics. Note that, even if different, the solutions of these two problems share the common property of being represented by sequences of jobs, i.e., by permutations of a given number of integers. Then in the Section 3 we illustrate the two hybrid metaheuristics considered, first introducing the main features of the pure population-based metaheuristics, respectively ACO and PSO, then showing how these are combined with SA, as well as discussing alternative triggering rules that can be used to determine the SA starting solutions. In the Section 4 we report the experimental test performed, comparing the obtained results with the ones of other algorithms from the literature. Finally, in the Section 5 we draw the chapter conclusions.

2. The referenced scheduling problems

In this section we briefly introduce the characteristics of the two scheduling problems faced by the two hybrid metaheuristics presented in the following, reporting also some literature review for them. These problems are the Single Machine Total Weighted Tardiness with Sequence-Dependent Setups (STWTSDS) problem and the Permutation Flowshop Scheduling (PFS) problem. Even if apparently different, the solutions to such problems have a common structure since they can both represented by permutation. For this reason, we introduced here some common notation. In general a solution x to one of the two scheduling problems involving a set of n jobs can be represented by a permutation or sequence $\sigma(x)=([1],..., [n])$, where $[j]$ indicates the index of the job sequenced in the j-th place. In addition we denote with $\varphi_\sigma:\{1,..., n\}\rightarrow\{1,..., n\}$, the mapping between the places in a sequence σ and the indexes of the sequenced jobs; for example, if job j is sequenced in the h-th place of σ we have $j=\varphi_\sigma(h)$.

2.1 The single machine total weighted tardiness problem with sequence-dependent setups

The STWTSDS problem consists in scheduling n independent jobs on a single machine. All the jobs are released simultaneously, i.e., they are ready at time zero, the machine is continuously available and it can process only one job at a time. For each job j=1,..., n, the following quantities are given: a processing time p_j, a due date d_j and a weight w_j. A sequence-dependent setup time s_{ij} must be waited before starting the processing of job j if it is immediately sequenced after job i. Setup operations are necessary to prepare production resources (e.g., machines) for the job to be executed next, and whenever they depend, as in this case, on the (type of) preceding job just completed they are called sequence-dependent setups. The tardiness of a job j is defined as $T_j=max(0, C_j-d_j)$, being C_j the job j completion time. The scheduling objective is the minimization of the total weighted tardiness expressed as $\sum_{j=1}^{n} w_j T_j$. This problem, denoted as $1/s_{ij}/\Sigma w_j T_j$, is strongly NP-hard since it is a special case of the $1//\Sigma w_j T_j$ that has been proven to be strongly NP-hard in (Lawler, 1997) (note that also the $1//\Sigma T_j$ special case is still NP-hard (Du & Leung, 1990)). Apart from its complexity, the choice of the STWTSDS as reference problem is also motivated by its relevance for manufacturing industries; in particular, the importance of performance criteria involving due dates, such as (weighted) total tardiness or total earliness and tardiness (E-T), as well as the explicit consideration of sequence-dependent setups, has been widely recognized in many real industrial contexts. In the literature both exact algorithms and heuristic algorithms have been proposed for the STWTSDS problem or for a slightly different version disregarding the job weights. However, since only instances of small dimensions can be solved by exact approaches, recent research efforts have been focused on the design of heuristics. The apparent tardiness cost with setups (ATCS) heuristic (Lee et al., 1997) is currently the best *constructive* approach for the STWTSDS problem. However, constructive heuristics, even if requiring smaller computational efforts, are generally outperformed by improvement, i.e., local search, and metaheuristics approaches. The effectiveness of stochastic search procedures for the STWTSDS is shown in (Cicirello & Smith, 2005), where the authors compare a value-biased stochastic sampling (VBSS), a VBSS with hill-climbing (VBSS-HC) and a simulated annealing (SA), to limited discrepancy search (LDS) and heuristic-biased stochastic sampling (HBSS) on a 120 benchmark problem instances for the STWTSDS problem defined by Cicirello in (Cicirello, 2003). The literature about applications of metaheuristics to scheduling is quite extended. In (Liao & Juan, 2007) an ACO algorithm for the STWTSDS is proposed, which is able to improve about 86% of the best known results for the Cicirello's benchmark previously found by stochastic search procedures in (Cicirello & Smith, 2005) Recently the Cicirello's best known solutions have been further independently improved in (Cicirello, 2006) by means of a GA approach, in (Lin & Ying, 2006) with three SA, GA and TS algorithms, in (Anghinolfi & Paolucci, 2008) using an ACO approach and in (Anghinolfi & Paolucci, 2007b) with PSO.

2.2 The permutation flowshop scheduling problem

The PFS problem requires to schedule a set of *n* jobs on a set of *m* machines so that each job is processed by each machine and the sequence of jobs is the same for all the machines. Then, a permutation of the *n* jobs identifies a solution to the PFS problem which consists in finding an optimal permutation for the jobs. For each job *j* and machine *h* the processing time p_{jh} is given; then, the completion times of the jobs on the machines can be computed for any given permutation σ=([1],..., [*n*]) of *n* jobs as follows

$$C_{[1]}^{1} = p_{[1],1} \tag{1}$$

$$C_{[j]}^{1} = C_{[j-1]}^{1} + p_{[j],1} \quad \forall j = 2,\dots,n \tag{2}$$

$$C_{[1]}^{h} = C_{[1]}^{h-1} + p_{[1],h} \quad \forall h = 2,\dots,m \tag{3}$$

$$C_{[j]}^{h} = \max\{C_{[j-1]}^{h}, C_{[j]}^{h-1}\} + p_{[j],h} \quad \forall h = 2,\dots,m;\ j = 2,\dots,n \tag{4}$$

where $C_{[j]}^{h}$ represents the completion time of the *j*-th job in the permutation on machine *h*. The scheduling problem is to find the job permutation σ* that minimizes the *makespan* $C_{\max}$, corresponding to the completion time of the last job on the *m*-th machine, i.e., $C_{\max} = C_{[n]}^{m}$. The makespan minimization for the PFS problem, denoted as $n/m/P/C_{\max}$, was originally proposed in (Johnson, 1954) and afterwards it has been widely investigated in the literature. This problem is NP-hard in the strong sense (Garey et al., 1976) for $m≥3$ and only instances of limited size can be solved by exact solution methods in an acceptable computation time. Therefore numerous heuristics approaches have been proposed in the literature, among which constructive heuristics (e.g., (Palmer, 1965), (Campbell et al., 1970), (Taillard, 1990)) improvement heuristics (e.g, (Ho & Chang, 1991), (Woo & Yim, 1998), (Suliman, 2000)) and metaheuristics as SA ((Osman & Potts, 1989), (Ishibuchi et al., 1995)), TS ((Nowicki & Smutnicki, 1996), (Grabowski and Wodecki, 2004)), GA ((Reeves, 1995), (Ruiz et al., 2006)), ACO ((Rajendran & Ziegler, 2004)) and PSO algorithms ((Liao et al., 2007), (Lian et al., 2006a), (Tasgetiren et al., 2007), (Jarboui et al., 2007)), some of which are taken as reference for the performance evaluation of the PSO-SA proposed in the following.

3. Two hybrid population-based metaheuristics

In this section we introduce the main concepts of ACO and PSO and we show how two hybrid algorithms, respectively ACO-SA and PSO-SA, can be derived through the interaction with SA. Note that in order to illustrate the specific characteristics of the algorithms we refer to the STWTSDS problem for ACO-SA and to the PFS one for PSO-SA.

3.1 The hybrid ant colony optimization algorithm

The ACO metaheuristic aims at exploiting the successful behaviour of real ants in cooperating to find shortest paths to food for solving combinatorial problems (Dorigo & Stützle, 2002), (Dorigo & Blum, 2005). Most of the real ants use *stigmergy* during food search, i.e., they have an effective indirect way to communicate each other which is the most promising trail, and finally the optimal one, towards food. Ants produce a natural essence, called pheromone, which is left on the followed path to food in order to mark it. The pheromone trail evaporates over time, finally disappearing on the abandoned paths. On the other hand, the pheromone trail can be reinforced by the passage of further ants; due to this fact effective (i.e., shortest) paths leading to food are finally characterized by a strong pheromone trail, and they are followed by most of ants. The ACO metaheuristic was first introduced in (Dorigo et al., 1991), (Dorigo et al., 1996) and (Dorigo, 1992), and since then it has been the subject of both theoretical studies and applications. ACO combines both *Reinforcement Learning* (RL) (Sutton & Barto, 1998) and *Swarm Intelligence* (SI) (Kennedy & Eberhart, 2001) concepts:

- each single agent (an ant) takes decisions and receives a reward from the environment, so that the agent's policy aims at maximizing the cumulative reward received (RL);
- the agents exchange information to share experiences and the performance of the overall system (the ant colony) emerges from the collection of the simple agents' interactions and actions (SI).

ACO has been successfully applied to several combinatorial optimization problems, from the first travelling salesman problem applications (Dorigo et al., 1991), (Dorigo et al., 1996), to vehicle routing problems (Bullnheimer et al., 1999), (Reinmann et al., 2004), and to single machine and flow shop scheduling problems (den Besten et al., 2000), (Gagné et al., 2002) and (Ying & Liao, 2004).

In this section we present a new hybrid ACO-SA approach to face the STWTSDS problem. In (Anghinolfi & Paolucci, 2008) we recently introduced the main characteristics of the pure ACO component of ACO-SA, which mainly differ from previous approaches in the literature for the following aspects: (a) we use a new pheromone trail model whose pheromone values are independent of the problem cost (or quality) function and they are bounded within an arbitrarily chosen and fixed interval; (b) we adopt a new global pheromone update (GPU) rule which makes the pheromone values asymptotically increase (decrease) towards the upper (lower) bound, without requiring any explicit cut-off as in the *Max-Min* Ant System (MMAS) (Stützle & Hoos, 2000); (c) we use a diversification strategy based on a temporary perturbation of the pheromone values performed by a local pheromone update (LPU) rule within any single iteration. The ACO that we proposed in (Anghinolfi & Paolucci, 2008) is mainly based on the Ant Colony System (ACS) (Dorigo & Gambardella, 1997), and it includes concepts inspired to the MMAS (Stützle & Hoos, 2000) and to the approaches in (Merkle & Middendorf, 2000), (Merkle & Middendorf, 2003), even if such concepts are encapsulated in a new pheromone model and exploited in a real different manner. We report in Figure 1 the very high level structure of the ACO-SA algorithm. In the following we will detail all the involved steps apart from *SA intensification* that we will describe in a separate subsection as this step is in common with the PSO-SA algorithm.

```
Initialization;
k=1;
While <termination condition not met>
{
  For each ant $a \in A$
  {
    Construction of solution $x_a^k$;
    Local pheromone update;
  }
  SA intensification;
  Global pheromone update;
  k=k+1;
}
```

Figure 1. The overall ACO-SA algorithm

We consider a set *A* of *na* artificial ants. At each iteration *k*, every ant *a* identifies a solution x_a^k building a sequence $\sigma(x_a^k)$ of the *n* jobs, whose objective value $Z(x_a^k)$ is then simply computed by executing each job at its feasible earliest start time for that sequence. Every ant *a* builds the sequence $\sigma(x_a^k)$ by iterating *n* selection stages: first, the set of not sequenced jobs for ant *a*, U_a^0, is initialized as $U_a^0 = \{1,...,n\}$; then, at stage *h*=1,..., *n*, the ant *a* selects one job *j* from the set U_a^{h-1} to be inserted in the position *h* of the partial sequence, and updates $U_a^h = U_a^{h-1} \setminus \{j\}$; at stage *h*=*n* all the jobs are sequenced and $U_a^n = \emptyset$. The job selection at each stage *h* of the construction procedure at iteration *k* is based on a rule that is influenced by the pheromone trail $\tau_k(h,j)$ associated with the possible *solution components*, i.e., position-job pairs, (*h, j*), where $j \in U_a^{h-1}$. Differently from other approaches in the literature, the pheromone values assigned to $\tau_k(h,j)$ are independent of the objective or quality function values associated with previously explored solutions including the component (*h, j*). In particular, we adopt an arbitrary range $[\tau_{Min}, \tau_{Max}]$ for the pheromone values, which is independent of the specific problem or instance considered; therefore any pair of values, such that $\tau_{Min} < \tau_{Max}$, can be chosen so that τ_{Max} and τ_{Min} are not included in the set of parameters that must be specified for the algorithm. In addition, the GPU rule controlling the ant colony learning mechanism imposes a smooth variation of $\tau_k(h,j) \in [\tau_{Min}, \tau_{Max}]$ such that both the bounds are only asymptotically reached. Note that also in MMAS lower and upper bounds are imposed for $\tau_k(h,j)$, but they must be appropriately selected, dynamically updated each time a new best solution is found, taking into account the objective function values, and they are used as cut-off thresholds. In the following we consider relative pheromone values $\tau'_k(h,j) = \tau_k(h,j) - \tau_{Min}$ such that $\tau'_k(h,j) \in [0, \tau'_{Max}]$, where $\tau'_{Max} = \tau_{Max} - \tau_{Min}$, whenever this makes simpler and more readable the expressions introduced.

Initialization. For each solution component (*h, j*), *h, j*=1,..., *n*, we assign an initial value of the pheromone trail by fixing $\tau_0(h,j) = (\tau_{Max} + \tau_{Min})/2$; in addition, we initialize the best current solution x^* as an empty solution, fixing the associated objective value $Z(x^*)$ to infinity.

Job selection rule. At a selection stage h of iteration k, an ant a determines which job $j \in U_a^{h-1}$ is inserted in the h-th position of the sequence as follows. First, similarly to the ACS, the ant chooses which job selection rule to use between *exploitation* and *exploration*: a random number q is extracted from the uniform distribution $U[0, 1]$ and if $q \le q_0$ the ant uses the exploitation rule, otherwise the exploration one. The parameter q_0 (fixed such that $0 \le q_0 \le 1$) directs the ants' behaviour towards either the exploration of new paths or the exploitation of the best paths previously emerged. The *exploitation* rule selects the job j in a deterministic way as

$$j = \arg\max_{u \in U_a^{h-1}} \{\tau'_k(h, j) \cdot [\eta(h, j)]^\beta\} \tag{5}$$

whereas the *exploration* rule according to a *selection probability* $p(h, j)$ computed as

$$p(h, j) = \frac{\tau'_k(h, j) \cdot [\eta(h, j)]^\beta}{\sum_{u \in U_a^{h-1}} \tau'_k(h, j) \cdot [\eta(h, j)]^\beta} \tag{6}$$

The quantity $\eta(h, j)$, associated with the solution component (h, j), is an heuristic value computed equal to the priority $I_t(h, j)$ of assigning job j in position h at time t according to the ATCS rule (Lee et al., 1997)

$$\eta(h, j) = I_t(h, j) = \frac{w_j}{p_j} \exp\left[\frac{\max(d_j - p_j - t, 0)}{k_1 \bar{p}}\right] \exp\left[-\frac{s_{\varphi_\sigma(h-1) j}}{k_2 \bar{s}}\right] \tag{7}$$

where

$$t = \sum_{i=1}^{h-1} (s_{\varphi_\sigma(i-1)\varphi_\sigma(i)} + p_{\varphi_\sigma(i)}) + s_{\varphi_\sigma(h-1) j} \tag{8}$$

$\bar{p}$ and $\bar{s}$ are respectively the average processing time and the average setup time, and k_1 and k_2 are the lookahead parameters fixed as originally suggested in (Lee et al., 1997). Therefore, in the ACO-SA algorithm the influence of the sequence-dependent setups is encapsulated in the heuristic values used in the job selection rule. The parameter β in (5) and (6) is the relative importance of the heuristic value with respect to the pheromone trail one.

Local pheromone update (*intra-iteration diversification*). As often done in ACO approaches to avoid premature convergence of the algorithm, a LPU is performed after any single ant a completed the construction of a solution x_a in order to make more unlike the selection of the same sequence by the following ants. In the ACO-SA we adopt the following the local pheromone update rule

$$\tau'_k(h, j) = (1 - \rho) \cdot \tau'_k(h, j) \quad \forall h = 1, \ldots, n;\ j = \phi_\sigma(h) \tag{9}$$

where ρ is a parameter fixed in [0, 1]. We must remark that such kind of update strictly *local*, i.e., we use it to favour the diversification of the sequences produced by the ants within the same iteration and (9) temporarily modifies the pheromone values only in the single iteration scope, since such changes are deleted before executing the *global pheromone update phase* and starting the next iteration. We denoted in (Anghinolfi & Paolucci, 2008) this feature is as *reset of the local pheromone update* (RLPU).

Global pheromone update. The (relative) pheromone values $\tau'_k\,(h, j)$ are varied within the range $[0, \tau'_{Max}]$ during the GPU phase with a rule, called *Unbiased Pheromone Update* (UPU), that we introduced in (Anghinolfi & Paolucci, 2008). The UPU rule does not uses cost or quality function values, but smoothly updates of pheromone trails associated with a set of quality solution components. We denote with Ω_k^* the *best component set* determined after the completion of iteration *k*; then, the UPU rule consists of the three following steps:

1. pheromone evaporation for the solution components not included in Ω_k^*

$$\tau'_{k+1}(h,j) = (1-\alpha)\cdot\tau'_k(h,j) \quad \forall(h,j)\notin\Omega_k^* \tag{10}$$

   where $0 \le \alpha \le 1$ is a parameter establishing the evaporation rate;

2. computation of the maximum pheromone reinforcement $\Delta\tau'_k(h,j)$ for the solution components in Ω_k^*

$$\Delta\tau'_k(h,j) = \tau'_{Max} - \tau'_k(h,j) \quad \forall(h,j)\in\Omega_k^* \tag{11}$$

3. update of the pheromone trails to be used in the next iteration for the solution components in Ω_k^*

$$\tau'_{k+1}(h,j) = \tau'_k(h,j) + \alpha\cdot\Delta\tau'_k(h,j) \quad \forall(h,j)\in\Omega_k^* \tag{12}$$

The UPU rule guarantees that $\tau'_k(h,j)\in[0,\tau'_{Max}]$ and that $\tau'_k(h,j)$ converges towards the bounds asymptotically ($\Delta\tau'_k(h,j)$ is progressively reduced as much as $\tau'_k(h,j)$ approaches to τ'_{Max}, as well as the decrease of $\tau'_k(h,j)$ towards 0 in (10)) with a law similar to the most frequently used cooling schedule for SA. The set Ω_k^* adopted in the ACO-SA is the one defined in (Anghinolfi & Paolucci, 2008) as the *Best-so-far* (BS) solution component set, that is, it includes only the solution components associated with the best sequence σ^* find so far

$$\Omega_k^* = \{\,(h,j): h=1,\dots,n;\ j=\phi_{\sigma^*}(h)\,\} \tag{13}$$

Termination conditions. The algorithm is stopped when a maximum number of iterations, or a maximum number of iterations without improvements, is reached.

3.2 The hybrid particle swarm optimization algorithm

PSO is a recent metaheuristic approach motivated by the observation of the social behaviour of composed organisms, such as bird flocking and fish schooling, and it tries to exploit the concept that the knowledge to drive the search for optimum is amplified by social interaction. PSO executes a population-based search in which the exploring agents, the *particles*, modify their positions during time according not only to their own experience, but also to the experience of other particles. In particular, a particle *p* may change its position with a velocity that in general includes a component moving *p* towards the best position so far achieved by *p* to take into account the particle experience, and a component moving *p* towards the best solution so far achieved by any among a set of neighbouring particles (*local neighbourhood*) or by any of the exploring particles (*global neighbourhood*). Note that, differently from GA, the PSO population is maintained and not filtered. PSO is based on the *Swarm Intelligence* (SI) concept (Kennedy & Eberhart, 2001): the agents are able to exchange information in order to share experiences, and the performance of the overall multi-agent system (the swarm) emerges from the collection of the simple agents' interactions and actions. PSO has been originally developed for continuous nonlinear optimization (Kennedy & Eberhart, 1995), (Abraham et al., 2006). The basic algorithm for a global optimization problem, corresponding to the minimization of a real objective function $f(x)$ of a variable vector x defined on a n-dimensional space, uses a population (*swarm*) of np particles; each particle i of the swarm is associated with a position in the continuous n-dimensional search space, $x_i=(x_{i1},\ldots, x_{in})$ and with the correspondent objective value $f(x_i)$ (*fitness*). For each particle i, the best previous position, i.e. the one where the particle found the lowest objective value (*personal best*), and the last particle position change (*velocity*) are recorded and represented respectively as $p_i=(p_{i1},\ldots, p_{in})$ and $v_i=(v_{i1},\ldots, v_{in})$. The position associated with the current smallest function value is denoted as $g=(g_1,\ldots, g_n)$ (*global best*). Denoting with x_i^k and v_i^k respectively the position and velocity of particle i at iteration k of the PSO algorithm, the following equations are usually used to iteratively modify the particles' velocities and positions:

$$v_i^{k+1} = w \cdot v_i^k + c_1 r_1 \cdot (p_i - x_i^k) + c_2 r_2 \cdot (g - x_i^k) \tag{14}$$

$$x_i^{k+1} = x_i^k + v_i^{k+1} \tag{15}$$

where w is the *inertia* parameter that weights the previous particle's velocity; c_1 and c_2, respectively called *cognitive* and *social* parameter, multiplied by two random numbers r_1 and r_2 uniformly distributed in [0, 1], weight the velocity towards the particle's personal best, $(p_i - x_i^k)$, and the velocity towards the global best solution, $(g - x_i^k)$, found so far by the whole swarm. The new particle position is determined in (15) by adding to the particle's current position the new velocity computed in (14). The PSO velocity model given by (14) and (15) is called *gbest*, but also a *lbest* model is introduced in (Kennedy & Eberhart, 2001): in this latter model the information about the global best position found so far by the whole group of particles is replaced by the local best position for each particle i, $l_i=(l_{i1},\ldots,l_{in})$, i.e., the position of the best particle found so far among a subset of particles nearest to i. The PSO parameters that we need to fix are the inertia w, the cognitive and social parameters c_1 and c_2, and finally the dimension of the swarm np; taking into account that in the standard PSO for continuous optimization $c_1+c_2=4.1$ (Clerc & Kennedy, 2002), the number of parameters needed by this metaheuristic is quite reduced.

In recent years many there is an increasing attention in the literature for application of the PSO approach to discrete combinatorial optimization problems. For example, PSO has been applied to the traveling salesman problem (TSP) (Pang et al., 2004), the vehicle routing problem (Chen et al., 2006), and scheduling problems (Tasgetiren et al., 2004), (Liao et al.,

2007), (Lian et al., 2006a), (Lian et al., 2006b), (Allahverdi & Al-Anzi, 2006) and (Parsopoulos & Vrahatis, 2006). Discrete PSO (DPSO) approaches differ both for the way they associate a particle position with a discrete solution and for the velocity model used; in particular, since here we consider a combinatorial problem whose solutions are represented by permutations, we could classify the DPSO approaches in the literature according to three kinds of solution-particle mapping, i.e., binary, real-valued and permutation-based, and three kinds of velocity model used, i.e., real-valued, stochastic or based on a list of moves. The first DPSO algorithm proposed in (Kennedy & Eberhart, 1997) used a binary solution representation and a stochastic velocity model since it associates the particles with *n*-dimensional binary variables and the velocity with the probability for each binary dimension to take value one. In (Tasgetiren et al., 2007), (Tasgetiren et al., 2004), (Parsopoulos & Vrahatis, 2006) real values are associated with the particle dimensions to represent the job place in the scheduling sequence according to a *random key representation* (Bean, 1994), and a *smallest position value* (SPV) rule is exploited to transform the particle positions into job permutations. Permutation-based solution-particle mappings are used in (Hu et al., 2003) for the *n*-queens problem together with a stochastic velocity model, representing the probability of swapping items between two permutation places, and a mutation operator, consisting of a random swap executed whenever a particle coincides with the *local* (*global*) *best* one. In (Lian et al., 2006a) particles are associated with job sequences and velocities are implemented as *crossover* and *mutation* operators borrowed from the genetic algorithm approach. Generally the velocity models adopted in DPSO approaches are either stochastic or real-valued. To our best knowledge the unique examples of velocity models based on a list of moves can be found in the DPSO approach for the TSP in (Clerc, 2004), together with the new DPSO approach that we very recently presented in (Anghinolfi & Paolucci, 2007b) to face the STWTSDS problem. This velocity model is quite difficult to be used as it needs the definition of an appropriate set of operators to extend the PSO computations in a discrete solution space.
In the following we illustrate the main features of the hybrid PSO-SA which extends the DPSO approach introduced in (Anghinolfi & Paolucci, 2007b) to face the PFS problem. As for the algorithm in (Anghinolfi & Paolucci, 2007b), PSO-SA is based on both a permutation solution-particle representation and on a list-of-moves velocity model, but differently we here introduce a new restart mechanism to avoid the stagnation of particles. In Figure 2 we report the overall structure of the PSO-SA algorithm. Then, similarly to what done for ACO-SA, we will detail the main PSO steps, finally dealing with the *SA intensification* in the last subsection.

```
Initialization;
While <termination condition not met>
{
  For each particle
  {
    Velocity update;
    Position update;
    Fitness computation;
  }
  SA intensification;
  Group restart;
  Best references update;
}
```
Figure 2. The PSO-SA algorithm.

We use a set of *np* particles, each one associated with a permutation σ, that is, with a schedule x whose fitness is given by the cost value $Z(x)$. To define the particle behaviour we need to introduce a metric for the permutation space and a set of operators to compute velocities and to update particles' positions consistently. As illustrated in (Anghinolfi & Paolucci, 2007b) we define a velocity as a set of moves, called *pseudo-insertion* (PI) moves, that, if applied to a given particle or permutation, change the position of the particle, generating a different permutation. Velocities are produced as difference between particle positions. For example, given a pair of particles p and q, the velocity v moving particle p from its current position to the one of particle q is a list of PI moves computed as the difference $v=\sigma_q-\sigma_p$. A PI move is a pair (j, d), where d is an integer displacement that is applied to job j within the permutation. Assuming for example that $j=\varphi(h)$, a PI move (j, d), which delays a job j in the permutation σ, extracts j from its current place h in σ and reinserts it generating a new permutation such that $j=\varphi(\min(h+d, n))$; analogously, a PI move $(j, -d)$, which instead anticipates a job j, produces a new sequence such that $j=\varphi(\max(h-d, 0))$. If for example we consider two particles associated with two permutations of $n=4$ jobs, $\sigma_p=(1,2,3,4)$ and $\sigma_q=(2,3,1,4)$, then, we compute the velocity $v=\sigma_q-\sigma_p=\{(1,2),(2, 1),(3,-1)\}$. The list of PI moves representing a velocity can include at most a single PI move for a given job.

We define a *position-velocity sum* operator to change the particle positions in the permutation space, which applies the PI moves included in a velocity list one at a time by extracting the involved job from the permutation and reinserting it in the new place. We call these moves as pseudo-insertion since in general they do not produce feasible permutations but what we called *pseudo-permutations*. We illustrate this point with an example: if we apply to the permutation $\sigma_p=(1,2,3,4)$ the first move in the velocity $v=\{(1,2),(2, -1),(3,-1)\}$, then we extract job 1 from the first place and reinsert it in the third place obtaining the pseudo-permutation (-,2,[3,1],4), where symbol "-" denotes that no job is now assigned to the first place, whereas [3,1] represents the ordered list of the two jobs 3 and 1 both assigned to the third place. Hence, PI moves produce in general not feasible permutations but pseudo-permutations characterized by one or more empty places and by others places containing a list of jobs. Then, we introduce the *permutation completion procedure* reported in Figure 3 to transform a pseudo-permutation into a feasible permutation. In Figure 3 $\pi(h)$ denotes the ordered set of items in the h-th place of the pseudo-permutation π, *pull*(s) the function that extracts the first element from an ordered set s, and *push*(i, s) the function that inserts the element i at the bottom of the set s. Hence, the permutation completion procedure manages $\pi(h)$ as a first-in-first-out (FIFO) list. As an example, starting from the pseudo-permutation π=([1,3],-,-,[4,2]) the permutation completion procedure produces the feasible permutation (3,1,4,2).

We define a *velocity sum* operator $\oplus$ which generates the list of PI moves for a velocity $w=v\oplus v'$ from the union of the PI moves in v and v'; in addition, since any job can appear only once in the PI list associated with a velocity, if v and v' include respectively the PI moves (j, d) and (j, d'), then w must include $(j, d+d')$ only if $d+d'\neq 0$. Finally, we define the *constant-velocity multiplication* so that the velocity computed as $w=c\cdot v$, where c is a real positive constant, includes the same PI moves of v whose displacement values have been multiplied by c.

```
Input: π a pseudo-sequence
Output: σ a feasible sequence
for each h=1,...,n
  {
    if |π(h)|=1 skip;
    else if |π(h)|=0
     {
       repeat
             k=h+1;
         while k<n and |π(k)|=0
         π(h)=pull(π(k));
     }
    else if |π(h)|>1
     {
         while |π(h)|>1
            push(pull(π(h), π(h+1));
        }
   }
σ=π;
```

Figure 3. The sequence completion procedure.

We can now consider the main steps of PSO-SA.

Initialization. A set of initial solutions, i=1,..., np, is assigned to the np particles by randomly generating a set of np permutations. This initialization procedure is similar to the one adopted for the discrete PSO approach in (Tasgetiren et al., 2007). Analogously, a set of np initial velocities is randomly produced and associated with the particles. In particular these velocities are generated first randomly extracting the number of PI moves composing a velocity from the discrete uniform distribution $U[[\lfloor n/4 \rfloor, \lfloor n/2 \rfloor]$, then, for each move, randomly generating the involved job and the integer displacement are respectively from $U[1, n]$ and from $U[\lfloor -n/3 \rfloor, \lfloor n/3 \rfloor]$. The set of particles is partitioned into n_c clusters G_{cl}, cl=1,..., n_c, randomly associating each particle to one of them, and the *local best* position l_i (i.e., the related solution x_{li}), computed as $l_i = \arg\min_{j \in G_{cl}} Z(x_j^0)$, is associated with each particle $i \in G_{cl}$. The quantity n_c is an input parameter of the algorithm. Finally, the global best position, that is the position associated with the best permutation found by any of the particles, is denoted with g (whose related solution is x_g).

Velocity and position update. At iteration k, we define for each particle i three velocity components, inertial (iv), directed to local best velocity (lv), and directed to global best velocity (gv), as follows:

$$iv_i^k = w \cdot v_i^{k-1} \tag{16}$$

$$lv_i^k = c_1 r_1 \cdot (l_i - \sigma_i^{k-1}) \tag{17}$$

$$gv_i^k = c_2 r_2 \cdot (g - \sigma_i^{k-1}) \tag{18}$$

Parameters w, c_1 and c_2 respectively represent the *inertia* parameter that weights the previous particle's velocity, and two kinds of *social* parameters, multiplied by two random numbers r_1 and r_2 extracted from $U[0,1]$, weighting the velocities towards the best position in the clusters (*local best*) and the *global best* position of the whole set of particles. Then, we update the particles' velocities by summing the three components (16), (17) and (18). The velocity model adopted for the PFS problem is the one called *glbest* in (Anghinolfi & Paolucci, 2007b) that does not include any velocity component directed towards the particle's personal best solution. In addition, differently from the standard PSO procedure, we compute the new position separately summing to the current particle position the three velocity components (16), (17) and (18) one at a time, so moving the particle through a set of intermediate feasible permutations obtained by the permutation completion procedure.

Restart of a group of particles. Differently from the DPSO in (Anghinolfi & Paolucci, 2007b), we restart all the particles in a group to avoid a premature convergence of the algorithm due to the stagnation of all the particles in one single position of the permutation space and to differentiate exploration. In particular, the positions of the particles belonging to the group whose local best solution is coincident with the global best solution of the swarm are reinitialized with a random solution and the local best is reset. Moreover, after such a reset, for the same group of particles we substitute for r iterations the weight of the global best velocity component c_2 with the value c_2^k computed according to the following rule

$$c_2^k = c_2\left(\frac{k-k'}{r}\right) \qquad k = k',\ldots k'+r \tag{19}$$

Since k' is the iteration at which the reset of the positions takes place and r is a parameter to be fixed, (19) corresponds to set for all the involved particles the value of the weight c_2 to 0 and then to make it linearly increase to its original value in r iterations. In this way the diversification effect of this group restart is reinforced since the particles in this group are not driven to immediately follow the direction towards the global best position but they can search for other good solutions independently.

3.3 The SA intensification

The *SA intensification* step included in the overall structures of both the ACO-SA and PSO-SA algorithms respectively in Figure 1 and 2 is performed using a SA procedure similar to the one adopted for the H-CPSO algorithm presented in (Jarboui et al., 2007). The SA algorithm, which originally took its inspiration from the simulation of the physical annealing of melted metals (Kirkpatric et al., 1983), iterates exploring the neighbourhood $N(x)$ of the current solution x accepting a stochastically generated candidate $x'\in N(x)$ with the following probabilistic rule: if $\Delta Z=Z(x)-Z(x')\leq 0$ then x' becomes the new current solution, otherwise x' is randomly accepted according to the probability distribution $P(\Delta Z, T)= e^{(\frac{-\Delta Z}{T})}$, where T is a control parameter playing the role of the temperature in the physical annealing process. This algorithm is usually initialized with a high value T_0 of the control parameter and it iterates progressively decreasing it until a termination condition is met according to a rule, called *cooling schedule,* which is critical for the algorithm convergence (Kirkpatric et al., 1983). In both the proposed hybrid algorithms to update T we adopt the exponential rule $T_k=T_0\cdot\theta^k$, where θ is a constant positive parameter. Similarly to (Jarboui et al., 2007) we use a stochastic definition of the neighbourhood $N(x)$ of the current solution x based on the random selection of insert and swap moves. In particular, we apply either an insert or a swap move on the permutation associated with x to generate the solution candidates at each SA iteration: first the algorithm randomly chooses with probability 0.5 between the two kinds of move, then it randomly selects the job to be moved and its insertion point or the pair of jobs to be swapped. The SA terminates when it reaches a maximum number of non improving iterations.

An important aspect to be considered whenever we embed an intensification procedure into a main metaheuristic is when such procedure is fired and which triggering rule is used. Designing a (hybrid) metaheuristic we should find an acceptable balance between exploration thoroughness and computational burden. Apparently, intensification steps greatly improve the accuracy of the search but also increase the time of computation. A quite straightforward choice for the two algorithms considered in this chapter is to perform intensification after all the exploring agents complete an iteration. Then, in ACO-SA the SA intensification takes place after all the ants generate a solution and in PSO-SA after all the particles have updated their position. Triggering rules specify which set X_{SA} of solutions in the current population have to be intensified, i.e., which solutions are chosen as starting point of SA. Even in this case a balance between accuracy and computation workload must be usually found. We can adopt rules selecting one or more starting points for intensification as detailed in the following.

a) The *best in iteration* (BI) rule: the SA is started from the (single) best solution found by the ants (particles) in the current iteration, i.e., $X_{SA} = \{x_{i^*}^k : i^* = \arg\min_{i=1,\dots,na} Z(x_i^k)\}$.
b) The *random* (RND) rule: the SA is started from a single solution that is randomly extracted from the solutions determined by the ants or particles in the current iteration k.
c) The *improved solution without intensification* (ISWI) rule: to implement this rule we need to define x_{WI}^* as the best solution found by any ant (particle) in the previous iterations without using the SA intensification. Then, the set X_{SA} may include one or more solutions found in the current iteration k improving x_{WI}^*, i.e. $X_{SA} = \{x_i^k : Z(x_i^k) < Z(x_{WLS}^*), i = 1,\dots,na\}$. Apparently, the number of solutions that can be subject to intensification at the end of an iteration with this rule can vary from zero to the number of ants *na* (or to the number of particles, *np*), even if the upper bound appear very unlikely.
d) The *all* (ALL) rule: the intensification is started from all the solutions found by the ants or particles in the current iteration k.

Independently of the used rule, if the solution produced by SA improves the starting $x_{i^*}^k$, then in ACO-SA the new solution may become the new current best and their relevant pheromone trails are updated accordingly, whereas in PSO-SA the new solution is associated with the particle *i**, so updating its relevant position, and the *lbest* solution for the cluster including particle *i** , as well as the *gbest* solution are possibly updated.

The BI and RND rules clearly outperform the ALL rule, and they are very likely to outperform also the ISWI one, under the computational time aspect as they both intensify a single solution. The ALL rule apparently should allow to produce solutions with the same quality of the other rules (we must keep in mind that intensification is executed with a stochastic algorithm) if we grant it a sufficiently long computation time, since it is a superset of them; on the other hand the ALL computational requirement is so high to make such rule hard to be accepted. Our experience also pointed out that the quality of the solutions yielded using RND are on the average dominated by the ones form the BI one. Therefore, we believe that in general the BI and ISWI may represent good compromise choices for triggering rules: the decision between these two rules can finally depend on the different time and quality requirements of the case under concern.

4. Experimental results

In this section we present some experimental results with the purpose of providing evidence on the possible benefit of combining SA with the two population-based metaheuristics considered. To this end we compared the behaviour of ACO-SA and PSO-SA with the one of the two same algorithms when LS is used instead of SA as intensification component. In particular, we adopted the deterministic LS procedure, reported in Figure 4, that, similarly to the SA algorithm described in the previous section, explores a mixed type of solution neighbourhood obtained by insert and swap moves.

```
x_b=x_c=x_0;
non_impr=0;
neigh_type=1;

repeat
  {
   x_c=x_b;

   x_c=best_in_neigh(x_b,neigh_type);

   if Z(x_c)<Z(x_b)
     {
      x_b=x_c;
      neigh_type=1;
     }
   else
     {
      non_impr++;
      neigh_type++;
     }
  }  until  (non_impr  >  max_non_impr)  and
neigh_type<=2;
```

Figure 4. The LS algorithm

We must observe that the LS in Figure 4 implements a kind of variable neighbourhood descent procedure (VND) (Hansen & Mladenovic, 1999), which for each current solution completely explores the neighbourhood generated by insert moves and, if no improvement is found, the one produced the swap moves. Then, in the following we report first the experimental tests performed for ACO-SA, giving greater emphasis to the analysis of the behaviour and relative effectiveness of the alternative triggering rules introduced in Section 3.3, whereas we limit the successive discussion on PSO-SA only on the comparison with the LS intensified version of algorithm. All the versions of the two algorithms analysed were coded in C++ and the experimental tests were executed on an Intel Core 2 6600, 2.4 GHz, 2 Gb PC (note however that our implementations do not exploit the dual processor architecture). During all the experimental campaign we adopted as termination criterion the maximum number of fitness (objective) function evaluations, that we fixed = 20,000,000. This choice follows the recommendation in (Taillard, 2005) suggesting the use of absolute computational burden measures (i.e., independent of the kind of computer) in order to obtain results easier to be compared in the scientific community. As regards the values of the parameters characterizing the SA procedure included in both the hybrid algorithms here considered, we fixed θ=0.95, the initial temperature $T_0 = -(0.2 \cdot Z_0)/\log(0.5)$ (such value is chosen to impose that at the initial iteration the probability of accepting a solution with a 20% deviation from objective value of the starting solution is 0.5), and imposing $10 \cdot n^2$ non improving iterations, where n is the number of jobs of the considered scheduling problem, as SA stopping criterion (note that similar settings are used in (Jarboui et al., 2007)).

4.1 The tests on ACO-SA

The benchmark that we adopted to analyse ACO-SA is the set of 120 problem instances for the STWTSDS with 60 jobs provided in (Cicirello, 2003) and available online at http://www.cs.drexel.edu/~cicirello/benchmarks.html. Note that this benchmark was used for testing various metaheuristic approaches recently appeared in the literature as (Cicirello & Smith, 2005), (Liao & Juan, 2007), (Cicirello, 2006), (Lin & Ying, 2006), (Anghinolfi & Paolucci, 2008) and (Anghinolfi & Paolucci, 2007b). The benchmark was produced by generating 10 instances for each combination of three different factors usually referenced in the literature (for a definition and discussion see, e.g., (Pinedo, 1995)): the due date tightness δ, the due date range R, and the setup time severity ξ, selected as follows: $\delta \in \{0.3, 0.6, 0.9\}$, $R \in \{0.25, 0.75\}$, $\xi \in \{0.25, 0.75\}$. For this set of tests we fixed the parameters characterizing the ACO as follows: na=30, α=0.09, β=0.5, ρ=0.05, q_0=0.7.
We conducted first a test in order to compare the possible triggering rules, i.e., BI, RND, ISWI and ALL, for ACO-SA. For each configuration of the algorithm c and for each instance i in the benchmark we executed 5 runs then computing the average result $\bar{Z}_{ci}$; after that, we obtained the best average result for each instance i as $\bar{Z}_i^* = \min_c \bar{Z}_{ci}$, and we computed for each configuration c and instance i the average percentage deviation Δ_{ci} from the best average $\bar{Z}_i^*$ as

$$\Delta_{ci} = \frac{\bar{Z}_{ci} - \bar{Z}_i^*}{\bar{Z}_i^*} \tag{20}$$

finally obtaining the overall average percentage deviation Δ_c for each configuration c as

$$\Delta_c = \frac{1}{I} \sum_{i=1}^{I} \Delta_{ci} \tag{21}$$

where I is the total number of instances considered. In Table 1 we summarise the obtained results. The columns of Table 1 report the overall average percentage deviations (Δ_c) and the relevant standard deviations (*Std*) for the four tested triggering rules, with and without the elimination of possible outliers; in fact, since in the objective values in the benchmark we observed differences of several orders of magnitude, the elimination of the outliers would reduce the possible influence of very slight absolute differences in the objectives for instances with small reference values. In particular, excluding the outliers we eliminated from the computation of the averages the instances with a percentage deviation not in the interval (-40%, 40%). In the last column of Table 1 we also show the average computational time (*CPU*) in seconds needed to terminate the runs.

			Without outliers (5 over 120)		
	Δ_c	**Std**	Δ_c	**Std**	**CPU (sec.)**
BI	0.14%	0.44%	0.10%	0.20%	23.8
RND	2.18%	6.10%	1.49%	4.89%	22.5
ISWI	7.51%	20.53%	4.21%	13.22%	20.8
ALL	7.99%	23.89%	4.41%	15.15%	20.6

Table 1. The comparison of intensification triggering rules for ACO-SA.

As we can observe, there are relevant differences both in the average percentage deviations and in the standard deviations for the tested rules. Then we executed the well-known non-parametric *Friedman's test* with 5% significance level obtaining that the differences between two groups of rules, one consisting of BI and RND, and the other ISWI and ALL, are significant from a statistical standpoint, both including and excluding the outliers. Therefore, at least for the kind of termination condition here considered, the two rules that execute a single SA intensification for iteration of the algorithm dominates the others. This may be due to the fact that the fixed maximum number of fitness function evaluations is better exploited by allowing fewer, here specifically one, SA search for iteration, so letting the whole algorithm execute a greater number of iterations. This behaviour is also suggested by the slightly larger computation time spent using the BI and RND configurations. Since we noted that the overall results for BI and RND in the first two rows of Table 1 were rather distant, we repeated the statistical test for a lower significance level, finding that the hypothesis that samples are not significantly different can be rejected when fixing a 4% level. In the second test performed we compared the ACO-SA results produced with the BI rule, with the one generated substituting SA with the LS described in Figure 4. In particular, we compared this latter configuration, denoted as ACO-LS, with ACO-SA in Table 2 (whose structure is analogous to Table 1).

			Without outliers (5 over 120)		
	Δ_c	**Std**	Δ_c	**Std**	**CPU (sec.)**
ACO-SA	0.15%	0.45%	0.11%	0.22%	23.8
ACO-LS	8.25%	22.33%	4.23%	10.05%	16.6

Table 2. The comparison of ACO-SA and ACO-LS.

The differences between the performance (both with and without outliers) of the two algorithms are apparent and also in this case their significance was confirmed by the statistical test with 5% significance level. Observing the computational times in the *CPU* column, we again could explain the worst behaviour of ACO-LS with the attitude of LS of being trapped in local optima: LS spent more fitness function evaluations than SA at each iteration as it deeply explores the basin of attraction of the intensified solution; then, the whole algorithm performed a smaller number of iterations. On the other hand, the ability of SA of escaping from local optima, i.e., its ability of diversifying the search, clearly turns out to be more effective.

Even if the evaluation of the performance of stochastic algorithms should always be based on average results, to complete the tests with the Cicirello's benchmark for the STWTSDS we report also the comparison of the best results over 5 runs obtained with ACO-SA and ACO-LS with a set of best known results. In particular, we consider an aggregate set of best known solutions combining the best solutions yielded by the following approaches: the ACO algorithm in (Liao & Juan, 2007), the GA in (Cicirello, 2006) and the SA, GA and TS algorithms in (Lin & Ying, 2006). Table 3 basically reproduces the same picture of Table 2, but here the possible presence of good solutions produced by chance for some instances should appear from the higher standard deviation values.

			Without outliers (8 over 120)	
	Δ_c	**Std**	Δ_c	**Std**
ACO-SA	1.18%	11.50%	0.99%	3.43%
ACO-LS	8.20%	21.74%	3.41%	6.04%

Table 3. The comparison of ACO-SA and ACO-LS with the best known solution.

4.2 The tests on PSO-SA

In order to evaluate the performance of PSO-SA compared to the one of the PSO algorithm with the LS presented in Figure 4 (denoted in the following as PSO-LS), we considered the well-known set of benchmark instances for the PFS problem with makespan criterion provided by Taillard (Taillard, 1993). In particular, we considered the benchmark set that includes 10 instances for n=20, 50, 100, 200, 500 jobs and m=5, 10, 20 machines (such classes of instances are denoted in the following with the n x m notation). For this test we used a set of np=2·n particles and a number of particle clusters n_c=np/10, fixing the values of the parameters needed by PSO as w=0.5, c_1=1 and c_2=2, setting r=40 for the instances with 20 and 50 jobs and r=20 for the ones with 100, 200 and 500 jobs. Similarly to the campaign for ACO-SA, we executed 5 runs for each benchmark instance, computing the average results, the best average results as previously described, finally the overall average percentage deviations as (21). In this case we directly compared PSO-SA and PSO-LS adopting BI as intensification firing rule. Table 4 summarizes the results produced by the two algorithms highlighting the outcomes for the different classes of instances as specified in the first column (*Problem*).

	Avg vs Avg (Δ_c)		Avg vs BK ($\Delta^{BK}{}_c$)		Total CPU		CPU for finding best	
Problem	**PSO-SA**	**PSO-LS**	**PSO-SA**	**PSO-LS**	**PSO-SA**	**PSO-LS**	**PSO-SA**	**PSO-LS**
20x5	0.00%	0.00%	0.04%	0.04%	23.5	26.5	0.1	0.2
20x10	0.00%	0.03%	0.00%	0.03%	41.2	42.0	3.7	4.4
20x20	0.01%	0.01%	0.02%	0.02%	76.5	79.0	16.7	18.3
50x5	0.00%	0.02%	0.00%	0.02%	39.8	39.5	3.2	4.0
50x10	0.00%	0.37%	0.54%	0.91%	79.3	71.7	29.7	29.8
50x20	0.00%	0.36%	1.04%	1.41%	149.3	154.9	78.4	77.9
100x5	0.02%	0.02%	0.11%	0.11%	76.9	65.8	17.2	17.7
100x10	0.03%	0.11%	0.70%	0.78%	146.7	126.3	55.9	42.9
100x20	0.03%	0.16%	2.41%	2.54%	242.3	275.0	138.7	185.1
200x10	0.00%	0.49%	0.16%	0.65%	253.6	212.7	105.8	106.2
200x20	0.00%	1.42%	1.34%	2.78%	377.2	462.8	270.3	416.8
500x20	0.00%	4.06%	0.75%	4.85%	900.0	1112.2	737.4	1104.2

Table 4. The comparison of PSO-SA with PSO-L1 for benchmark instance classes.

The first pair of columns in Table 4 reports the comparison between the overall average percentage deviations (Δ_c) from the best average; as it appears, PSO-SA outcomes are on the average never worse than the PSO-LS ones for each class of instances and also the Friedman's test with 5% significance level confirmed the statistical significance of this result. We must remark that we report here also the runs for the greatest instances with 500 jobs even if for such cases the value of maximum fitness function evaluations adopted as termination criterion turned out to be too restrictive: such value in fact allowed a too small number of iterations to really appreciate the behaviour of the whole hybrid approach (actually, we could consider the test for the 500x20 only a comparison between SA and LS). Nevertheless, we verified the statistical significance of the results even excluding the 500x20 instances. The second pair of columns in Table 4 shows the overall average percentage deviations ($\Delta^{BK}{}_c$) of the average PSO-SA and PSO-LS results from the best know solutions (BK) for the Taillard's PSP benchmark (we suggest the readers interested to BK to refer to Taillard's web site where this set is maintained and updated). The third pair of columns reports the total average CPU time needed by the compared algorithms to terminate, whereas the last pair of columns the average CPU needed to find the best solution produced in the runs (both values are in seconds). As we can observe, the differences among computational times are not really significant for this benchmark.
We show in Table 5 the overall comparison between PSO-SA and PSO-LS for the benchmark, including also the standard deviations.

	Avg vs Avg		Avg vs BK		CPU	
	Δ_c	Std	$\Delta^{BK}{}_c$	Std	Total	For best
PSO-SA	0.01%	0.03%	0.59%	0.73%	200.5	121.4
PSO-LS	0.59%	1.14%	1.18%	1.47%	222.4	167.3

Table 5. The overall comparison of PSO-SA and PSO-LS algorithms.

Finally, we can comment that also in the case of a hybrid PSO based algorithm, the presence of the SA search resulted apparently more effective than a LS one (we must recall that the LS used here has also a VND flavour), due to its powerful intensification ability but specifically to its attitude to smoothly diversify the exploration according to the reduction of the value of the parameter *T* ruled by the cooling schedule.

5. Conclusions

In this chapter we illustrated how SA can be exploited to embed in two alternative population-based metaheuristics a trajectory search component. Population-based metaheuristics need intensification procedures as LS to reach peak performances for discrete combinatorial problems. The effectiveness of using SA instead of LS to this end emerged from the experimental tests reported in this chapter. We considered ACO and PSO and we analysed the performance of the resulting hybrid algorithms on two scheduling problems quite extensively faced in the literature, the STWTSDS and the PSP problems. However, even different, the combinatorial structure of such problems is the same, as their relevant solutions can be represented by permutations. Actually, we compared two "structurally" similar trajectory methods, LS and SA: in particular we adopted a deterministic LS which explores a combination of two neighbourhoods generated respectively by insert and swap moves, with a VND fashion; similarly, the stochastic SA procedure at each iteration derives the next candidate solution first randomly selecting between an insert and a swap move. In other words we tried to use the same kind of ingredients in the two trajectory methods in order to measure their relative strength. Hence the results that we showed allow to conclude that the principles in SA can lead to superior solution improvement procedures than LS when the same level of sophistication is used in both of them, without implying the obviously wrong claim that "any" SA procedure is better than "any" LS.

Hybridization by combining a population-based algorithm, provided with memory, learning and/or swarm intelligence mechanisms, with SA is a viable strategy to produce in a simple way high quality metaheuristics. Therefore, we would recommend to consider also this possibility when tackling complex combinatorial problems: the intensification (the attitude of operating as a LS) and diversification (the attitude of not limit the search to a confined region) features that are blended in SA in a dynamic fashion (ruled by the cooling schedule) are certainly good ingredients for powerful hybrid methods.

6. References

Blum, C., Roli, A.: Metaheuristics in combinatorial optimization: Overview and conceptual comparison. ACM Computing Surveys 35(3) (2003) 268–308

Doerner, K.F.; Gendreau, M.; Greistorfer, P.; Gutjahr, W.; Hartl, R.F.; Reimann, M. (Eds.). Metaheuristics - Progress in Complex Systems Optimization. Springer. Series: Operations Research/Computer Science Interfaces Series , Vol. 39. 2007.

Raidl G.R. A unified view on hybrid metaheuristics. In Francisco Almeida et al., editors, Proceedings of the Hybrid Metaheuristics Workshop, volume 4030 of LNCS, pages 1-12. Springer, 2006.

Talbi, E.G.: A taxonomy of hybrid metaheuristics. Journal of Heuristics 8(5) (2002) 541–565.

Dorigo M., Stutzle T. Ant Colony Optimization. MIT Press. 2004.

Anghinolfi D., Paolucci M. Parallel machine total tardiness scheduling with a new hybrid metaheuristic approach. Computers & Operations Research, Volume 34, Issue 11, November 2007, Pages 3471-3490

Liu B, Wang L, Jin Y-H. An effective hybrid PSO-based algorithm for flow shop scheduling with limited buffers. Computers & Operations Research. 2008; 35: 2791-2806.

Qie He and Ling Wang. A hybrid particle swarm optimization with a feasibility-based rule for constrained optimization. Applied Mathematics and Computation, Volume 186, Issue 2, 15 March 2007, Pages 1407-1422.

Ling-lai Li, Ling Wang and Li-heng Liu. An effective hybrid PSOSA strategy for optimization and its application to parameter estimation. Applied Mathematics and Computation, Volume 179, Issue 1, 1 August 2006, Pages 135-146.

Ge, Hongwei Du, Wenli Qian, Feng A Hybrid Algorithm Based on Particle Swarm Optimization and Simulated Annealing for Job Shop Scheduling. Proceedings of ICNC 2007. Third International Conference on Natural Computation 2007. Volume: 3

Weijun Xia and Zhiming Wu. An effective hybrid optimization approach for multi-objective flexible job-shop scheduling problems. Computers & Industrial Engineering, Volume 48, Issue 2, March 2005, Pages 409-425

Nihan Çetin Demirel and M. Duran Toksarı. Optimization of the quadratic assignment problem using an ant colony algorithm. Applied Mathematics and Computation, Volume 183, Issue 1, 1 December 2006, Pages 427-435

Feng, Yuanjing; Feng, Zuren. Ant colony system hybridized with simulated annealing for flow-shop scheduling problems. WSEAS Transactions on Business and Econonomics. Vol. 1, no. 1, pp. 133-138. Jan. 2004.

Yogeswaran, M. Ponnambalam, S. G. Tiwari, M. K. An hybrid heuristic using genetic algorithm and simulated annealing algorithm to solve machine loading problem in FMS. Proc. of International Conference on Automation Science and Engineering, 2007. CASE 2007. IEEE On page(s): 182-187.

Lawler, E.L. (1997). A 'pseudopolynomial' algorithm for sequencing jobs to minimize total tardiness. Annals of Discrete Mathematics, 1: 331–342.

Du, J. and Leung, J.Y-T. (1990). Minimizing total tardiness on one machine is NP-hard. Mathematics of Operations Research, 15: 483–495.

Lee, Y.H., Bhaskaran, K. and Pinedo, M. (1997). A heuristic to minimize the total weighted tardiness with sequence-dependent setups. IIE Transaction, 29: 45-52.

Cicirello, V.A. and Smith S.F. (2005). Enhancing stochastic search performance by value-based randomization of heuristics. Journal of Heuristics, 11: 5–34.

Cicirello, V.A. (2003). Weighted tardiness scheduling with sequence-dependent setups: a benchmark library. Technical Report, Intelligent Coordination and Logistics Laboratory, Robotics Institute, Carnegie Mellon University, USA.

Liao C-J, Juan HC. An ant colony optimization for single-machine tardiness scheduling with sequence-dependent setups. Computers & Operations Research 2007; 34; 1899-1909.

Cicirello VA. Non-Wrapping Order Crossover: An Order Preserving Crossover Operator that Respects Absolute Position. In: Proceeding of GECCO'06 Conference, Seattle, Washington, USA; 2006. p. 1125-1131.

Lin S-W, Ying K-C. Solving single-machine total weighted tardiness problems with sequence-dependent setup times by meta-heuristics. The International Journal of Advanced Manufacturing Technology 2006; Available online (www.springerlink.com).

Anghinolfi D., Paolucci M., A new ant colony optimization approach for the single machine total weighted tardiness scheduling problem, International Journal of Operations Research, Vol. 5, No. 1, 44-60. 2008.

Anghinolfi D., Paolucci M., A new discrete particle swarm optimization approach for the single-machine total weighted tardiness scheduling problem with sequence-dependent setup times. European Journal of Operational Research (avaliable online) 2007.

Johnson SM. Optimal two-and three-stage production schedules. Naval Research Logistics Quarterly. 1954; 1: 61-68.

Garey MR, Johnson DS, Sethi R. The complexity of flowshop and jobshop scheduling. Mathematics of Operations Research. 1976; 1: 117-129.

Palmer DS. Sequencing jobs through a multistage process in the minimum total time: A quick method of obtaining a near-optimum. Operational Research Quarterly. 1965; 16: 101-107.

Campbell HG, Dudek RA, Smith ML. A heuristic algorithm for the n job, m machine sequencing problem. Management Science. 1970; 16(10): B630-B637.

Taillard E. Some efficient heuristic methods for the flowshop sequencing problems. European Journal of Operational Research. 1990; 47: 65-74.

Ho JC, Chang Y-L. A new heuristic for the n-job, m-machine flow-shop problem. European Journal of Operational Research. 1991; 52: 194-202.

Woo HS, Yim DS. A heuristic algorithm for mean flowtime objective in flowshop scheduling. Computers and Operations Research. 1998; 25: 175-182.

Suliman SMA. A two-phase heuristic approach to the permutation flow-shop scheduling problem. International Journal of Production Economics. 2000; 64: 143-152.

Osman I, Potts C. Simulated annealing for permutation flow shop scheduling. OMEGA. 1989; 17(6): 551-557.

Ishibuchi H, Misaki S, Tanaka H. Modified simulated annealing algorithms for the flow shop sequencing problem. European Journal of Operational Research. 1995; 81: 388-398.

Nowicki E, Smutnicki C. A fast tabu search algorithm for the permutation flowshop problem. European Journal of Operational Research. 1996; 91: 160-175.

Grabowski J, Wodecki M. A very fast tabu search algorithm for the permutation flowshop problem with makespan criterion. Computers and Operations Research. 2004; 31(11): 1891-1909.

Reeves C. A genetic algorithm for flowshop sequencing. Computers and Operations Research. 1995; 22(1): 5-13.

Ruiz R, Maroto C, Alcaraz J. Two new robust genetic algorithms for the flowshop scheduling problem. OMEGA. 2006; 34: 461-476.

Rajendran C, Ziegler H. Ant-colony algorithms for permutation flowshop scheduling to minimize makespan/total flowtime of jobs. European Journal of Operational Research. 2004; 155(2): 426-438.

Liao C-J, Tseng C-T, Luarn P. A discrete version of particle swarm optimization for flowshop scheduling problems. Computers & Operations Research. 2007; 34: 3099-3111.

Lian Z, Gu X, Jiao B: A similar particle swarm optimization algorithm for job-shop scheduling to minimize makespan. Applied Mathematics and Computation. 2006; 183: 1008-1017.

Tasgetiren MF, Liang Y-C, Sevkli M, Gencyilmaz G. A particle swarm optimization algorithm for makespan and total flowtime minimization in the permutation flowshop sequencing problem. European Journal of Operational Research. 2007; 177: 1930-1947.

Jarboui B, Ibrahim S, Siarry P, Rebai A. A combinatorial Particle Swarm Optimisation for solving permutation flowshop problems. Computers & Industrial Engineering. 2007; doi: 10.1016/j.cie. 2007.09.006.

Dorigo, M. and Stützle, T. (2002). The ant colony optimization metaheuristics: algorithms, applications and advances. In Handbooks of metaheuristics (Ed.: Glover, F. and Kochenberger, G). Int. Series in Operations Research & Management Science, Kluver, Dordrech, 57: 252-285.

Dorigo, M. and Blum, C. (2005). Ant colony optimization theory: A survey. Theoretical Computer Science, 344: 243-278.

Dorigo, M., Maniezzo, V. and Colorni, A. (1991). Positive feedback as a search strategy. Tech Report 91-016. Dipartimento di Elettronica, Politecnico di Milano, Italy.

Dorigo, M., Maniezzo, V. and Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. IEEE Trans. Systems, Man, Cybernet.-Part B, 26: 29-41.
Dorigo, M. (1992). Optimization, learning and natural algorithms (in Italian). PhD Thesis, Dipartimento di Elettronica, Politecnico di Milano, Italy.

Sutton, R.S. and Barto, A.G. (1998). Reinforcement Learning: An Introduction. MIT Press, Cambridge.

Kennedy, J. and Eberhart, R.C. (2001). Swarm Intelligence. Morgan Kaufmann Publishers.

Bullnheimer, B., Hartl, R.F. and Strauss, C. (1999). An improved ant system algorithm for the vehicle routing problem. Annals of Operations Research, 89: 319-328.

Reinmann, M., Doerner, K. and Hartl, R.F. (2004). D-ants: savings based ants divide and conquer the vehicle routing problems. Computers & Operations Research, 31(4): 563-591.

den Besten, M., Stützle, T. and Dorigo, M. (2000). Ant colony optimization for the total weighted tardiness problem. Proceeding PPSN VI, Sixth International Conference Parallel Problem Solving from Nature, Lecture Notes in Computer Science, Berlin, Springer, 1917: 611–20.

Gagné, C., Price, W.L. and Gravel, M. (2002). Comparing an ACO algorithm with other heuristics for the single machine scheduling problem with sequence-dependent setup times. Journal of the Operational Research Society, 53: 895–906.

Ying, G.C. and Liao, C.J. (2004). Ant colony system for permutation flow-shop sequencing. Computers & Operations Research, 31: 791–801.

Stützle, T. and Hoos, H.H. (2000). Max-min ant system. Future Generation Computer System, 16: 889–914.

Dorigo, M. and Gambardella, L.M. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. IEEE Transactions on Evolutionary Computation, 1: 53–66.

Merkle, D. and Middendorf, M. (2000). An ant algorithm with a new pheromone evaluation rule for total tardiness problems. Proceedings of the EvoWorkshops 2000, Springer Verlag LNCS 1803: 287–296.

Merkle, D. and Middendorf, M. (2003). Ant Colony Optimization with Global Pheromone Evaluation for Scheduling a Single Machine. Applied Intelligence, 18: 105–111.

Kennedy J, Eberhart R. Particle Swarm Optimization. Proceeding of the 1995 IEEE International Conference on Neural Network 1995; 1942-1948.

Abraham A, Guo H, Liu H. Swarm Intelligence: Foundations, Perspectives and Applications. In: Abraham A, Grosan C, Ramos V (Eds), Swarm Intelligence in Data Mining, Studies in Computational Intelligence (series). Springer-Verlag: Berlin; 2006.

Clerc M, Kennedy J. The particle swarm: Explosion, stability, and convergence in a multi-dimensional complex space. IEEE Transactions on Evolutionary Computation 2002; 6; 58-73.

Pang W, Wang KP, Zhou CG, Dong L-J. Fuzzy discrete particle swarm optimization for solving traveling salesman problem. In: Proceedings of the 4th International Conference on Computer and Information Technology. IEEE CS Press; 2004. p. 796 – 800.

Chen A, Yang G, Wu Z. Hybrid discrete particle swarm optimization algorithm for capacitated vehicle routing problem. Journal of Zhejiang Univ. SCIENCE A 2006; 7: 607-614.

Tasgetiren MF, Sevkli M, Liang YC, Gencyilmaz G. Particle swarm optimization algorithm for single machine total weighted tardiness problem. In: Proceedings of the IEEE congress on evolutionary computation, vol.2. Portland; 2004. p. 1412–1419.

Liao C-J, Tseng C-T, Luarn P. A discrete version of particle swarm optimization for flowshop scheduling problems. Computers & Operations Research 2007; 34; 3099-3111.

Lian Z, Gu X, Jiao B. A similar particle swarm optimization algorithm for permutation flowshop scheduling to minimize makespan. Applied Mathematics and Computation 2006a; 175; 773-785.

Lian Z, Gu X, Jiao B. A similar particle swarm optimization algorithm for job-shop scheduling to minimize makespan. Applied Mathematics and Computation 2006b; 183; 1008-1017.

Allahverdi A, Al-Anzi FS. A PSO and a Tabu search heuristics for the assembly scheduling problem of the two-stage distributed database application. Computers & Operations Research 2006; 33; 1056–1080.

Parsopoulos KE, Vrahatis MN. Studying the Performance of Unified Particle Swarm Optimization on the Single Machine Total Weighted Tardiness Problem, Lecture Notes in Artificial Intelligence (LNAI) 2006; 4304; 760-769.

Kennedy J, Eberhart R. A discrete binary version of the particle swarm algorithm. In: Proceedings of the International Conference on Systems, Man, and Cybernetics, vol.5. IEEE Press; 1997. p. 4104–4108.

Bean JC. Genetic algorithm and random keys for sequencing and optimization. ORSA Journal on Computing 1994; 6; 154-160.

Hu X, Eberhart R, Shi Y. Swarm intelligence for permutation optimization: a case study of n-queens problem. In: Proceedings of the 2003 IEEE Conference on Swarm Intelligence Symposium (SIS '03). IEEE Press; 2003. p. 243-246.

Clerc M. Discrete Particle Swarm Optimization. In: Onwubolu GC, Babu BV (Eds), New Optimization Techniques in Engineering. Springer-Verlag: Berlin; 2004; 219-240.

Kirkpatric S, Gelatt Jr. CD, Vecci MP. Optimization by simulated annealing. Science 1983; 220: 671-80.

Hansen P., Mladenovic N. Variable neighborhood search: Methods and recent applications. In Proceedings of MIC'99, pages 275–280, 1999.

Taillard E., Few guidelines for analyzing methods. in Tutorial, 6th Metaheuristics Int. Conf., 2005.
Pinedo M. Scheduling: Theory, Algorithms, and Systems. Prentice Hall: Englewood Cliffs, NJ; 1995.

Taillard E. Benchmarks for basic scheduling problems. European Journal of Operational Research. 1993; 64: 278-285.