26

Ant colonies for performance optimization of multi-components systems subject to random failures

Nabil Nahas, Mustapha Nourelfath and Daoud Ait-Kadi
Universite Laval, Mechanical Engineering Department CIRRELT
Canada

1. Introduction

Reliability has been considered as an important design measure in industry. The design of a system involves in general numerous discrete choices among available component types based on reliability, cost, performance, weight, etc. If the design objective is to maximize reliability for a certain cost requirement, then a strategy is required to identify the optimal combination of components and/or design configuration. This leads to combinatorial optimization problems which are NP-hard. For large industrial problems, exact methods are lacking since they require a very large amount of computation time to obtain the solution of the problem. This chapter will focus on the use of ant colonies to solve three optimal design problems which are among the most important in practice:

1. The reliability optimization of series systems with multiple-choice constraints incorporated at each subsystem, to maximize the system reliability subject to the system budget. This is a nonlinear binary integer programming problem and characterized as an NP-hard problem.
2. The redundancy allocation problem (RAP) of binary series-parallel systems. This is a well known NP-hard problem which involves the selection of elements and redundancy levels to maximize system reliability given various system-level constraints. As telecommunications and internet protocol networks, manufacturing and power systems are becoming more and more complex, while requiring short developments schedules and very high reliability, it is becoming increasingly important to develop efficient solutions to the RAP.
3. Buffers and machines selections in unreliable series-parallel production lines: we consider a series-parallel manufacturing production line, where redundant machines and in-process buffers are included to achieve a greater production rate. The objective is to maximize production rate subject to a total cost constraint. Machines and buffers are chosen from a list of products available in the market. The buffers are characterized by their cost and size. The machines are characterized by their cost, failure rate, repair rate and processing time. To estimate series-parallel production line performance, an analytical decomposition-type approximation is proposed. Simulation results show that this approximate technique is very accurate. The optimal design problem is formulated

Source: Swarm Intelligence: Focus on Ant and Particle Swarm Optimization, Book edited by: Felix T. S. Chan and Manoj Kumar Tiwari, ISBN 978-3-902613-09-7, pp. 532, December 2007, Itech Education and Publishing, Vienna, Austria

as a combinatorial optimization one where the decision variables are buffers and types of machines, as well as the number of redundant machines.

For each problem, a literature review will be presented. This review will focus on meta-heuristics in general, and on ant colonies in particular. Recent advances on efficient solution approaches based on Hybrid Ant Colony Optimization (HACO) will be detailed.

2. Reliability optimization of a series system

2.1. Literature review

As it is often desired to consider the practical design issue of handling a variety of different component types, this paper considers a reliability optimization problem with multiple-choice constraints incorporated. To deal with such reliability optimization problems with multiple-choice constraints incorporated, Sung and Cho [9] have used an efficient branch-and-bound method. Nourelfath and Nahas [6] have solved the reliability optimization problem by using quantized neural networks. [11] deals with a reliability optimization problem for a series system with multiple choice constraints incorporated to maximize the system reliability subject to the system budget. The problem is formulated as a binary integer programming problem with a non linear objective function [1], which is equivalent to a knapsack problem with multiple-choice constraints, so that it is NP-hard [3]. Some branch-and-bound methods for such knapsack problems with multiple-choice constraints have been suggested in the literature [5,7,8]. However, for large industrial problems, these methods are lacking since they require a very large amount of computation time to obtain the solution of the problem. This section describes the use of an *ant system* to obtain optimal or nearly optimal solutions very quickly.

2.2. Optimal design problem

Let us consider a series system of n components. For each component, there are different technologies available with varying costs, reliabilities, weights and other characteristics. The design problem we propose to study is to select the best combination of technologies to maximize reliability given cost. Only one technology will be adopted for each component. In order to formulate the problem in mathematical expression, the following notations are introduced first:

n	the number of components
Mi	the number of technologies available for the component i
C_i^j	the cost of a component i using the technology j (C_i^j is assumed to be known)
R_i^j	the reliability of the component i when the technology j is used
B	the available budget
TC	the total cost.

We specify the decision variable x_i^j (with $j = 1, 2, \ldots, M$; and $i = 1, 2, \ldots, n$) as:

$$x_i^j = \begin{cases} 1 \text{ if the component } i \text{ uses the technology } j \\ 0 \text{ otherwise} \end{cases}$$

Considering these notations, the proposed series-system reliability optimization problem is expressed in the following binary nonlinear integer programming problem:

Maximize $$Z = \prod_{i=1}^{n}\left(\sum_{j=1}^{M_i} x_i^j R_i^j\right)$$

Subject to $$\sum_{i=1}^{n}\sum_{j=1}^{M_i} x_i^j C_i^j \leq B \quad (1)$$

$$\sum_{j=1}^{M_i} x_i^j = 1 \ \forall \ i = 1, 2, \ldots, n \quad (2)$$

$$x_i^j = \{0, 1\} \ \forall \ j = 1, 2, \ldots, M_j \text{ and } i = 1, 2, \ldots, n \quad (3)$$

Constraint (1) represents the budget constraint; without loss of generality, we consider that B is an integer. Constraint (2) represents the multiple-choice constraint, and constraint (3) defines the decision variables.

When a solution satisfies all the constraints, it is called a feasible solution; otherwise, the solution is said to be infeasible. Our goal is to find an optimal solution or sometimes a nearly optimal solution. This is motivated by the fact that in real size industrial systems, the search space of the reliability optimization problem formulated in this paper is very large, taking the use on non heuristic approaches infeasible. Ant system is a recent kind of meta-heuristic which has been shown to be suitable (especially when combined with local search) for combinatorial optimization problems with a good neighborhood structure (see e.g. [6,10]), as in the case of the reliability optimization problem formulated in this paper.

2.3. Solution approach of the reliability optimization problem

To apply the ant system (AS) algorithm to a combinatorial optimization problem, it is convenient to represent the problem by a graph $G = (\varsigma, \Lambda)$, where ς are the nodes and Λ is the set of edges. To represent our problem as such a graph, the set of nodes g is given by components and technologies, and edges connect each component to its available technologies. Ants cooperate by using indirect form of communication mediated by pheromone they deposit on the edges of the graph G while building solutions.

Informally, our algorithm works as follows: m ants are initially positioned on the node representing the first component. Each ant will construct one possible structure of the entire system. In fact, each ant builds a feasible solution (called a tour) by repeatedly applying a stochastic greedy rule, called, the *state transition rule*. Once all ants have terminated their tour, the following steps are performed:

- An improvement procedure is applied. This procedure, which will be detailed later, is composed of a specific *improvement algorithm* (called algorithm 1) and a local search.
- The amount of pheromone is modified by applying the *global updating rule*.

Ants are guided, in building their tours, by both heuristic information and by pheromone information. Naturally, an edge with a high amount of pheromone is a very desirable choice. The pheromone updating rules are designed so that they tend to give more pheromone to edges which should be visited by ants.

The state transition rule used by the ant system is given in equation (4). This represents the probability with which ant k selects a technology j for component i:

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha . [\eta_{ij}]^\beta}{\sum_{m=1}^{M_i} [\tau_{im}(t)]^\alpha . [\eta_{im}]^\beta} \quad (4)$$

where τ_{ij} and η_{ij} are respectively the pheromone intensity and the heuristic information between component i and technology j . α is the relative importance of the trail and β is the relative importance of the heuristic information η_{ij}. The problem specific heuristic j information used is $\eta_{ij} = \frac{R_i^j}{C_i^j}$ where R_i^j and C_i^j represent the associated reliability and cost. That is, technologies with higher reliability and smaller cost have greater probability to be chosen.

During the construction process, no guarantee is given that an ant will construct a feasible solution which obeys the budget constraint (i.e. $\sum_{i=1}^{n}\sum_{j=1}^{M_i} x_i^j C_i^j \leq B$). The unfeasibility of solutions is treated in the pheromone update: the amount of pheromone deposited by an ant is set to a high value if the generated solution is feasible and to a low value if it is infeasible. These values are dependent of the solution quality. Infeasibilities can then be handled by assigning penalties proportional to the amount of budget violations. In the case of feasible solutions, an additional penalty proportional to the obtained solution is introduced to improve its quality.

Following the above remarks, the trail intensity is updated as follows:

$$\Delta\tau_{ij} = \sum_{k=1}^{m} \Delta\tau_{ij}^k \tag{5}$$

ρ is a coefficient such that (1 - ρ) represents the evaporation of trail and $\Delta\tau_{ij}$ is :

$$\tau_{ij}(new) = \rho\tau_{ij}(old) + \Delta\tau_{ij} \tag{6}$$

where m is the number of ants and $\Delta\tau_{ij}^k$ is given by:

$$\Delta\tau_{ij}^k = \begin{cases} Q.penalty_k & \text{if the k}^{\text{th}}\text{ ant chooses technology } j \text{ for component } i \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

where Q is a positive number, and $penalty_k$ is defined as follows:

$$penalty_k = \begin{cases} \left(\frac{B}{TC_k}\right)^a & \text{if } B < TC_k \\ \left(\frac{R_k}{R^*}\right)^b & \text{if } B \geq TC_k \end{cases} \tag{8}$$

B is the available budget, TCk is the total cost obtained by ant k, R_k is the reliability obtained by ant k and R^* is the best obtained solution. Parameters a and b represent the relative importance of penalties. It can be easily seen from the above equations that by introducing a penalty function, we aim at encouraging the AS algorithm to explore the feasible region and infeasible region that is near the border of feasible area and discouraging, but allowing, search further into infeasible region.

It is well known that the performance of AS algorithms can be greatly enhanced when coupled to local improvement procedures [2]. Following this, two local improvement algorithms are included in our AS approach (called local search algorithm and algorithm 1). Algorithm 1 uses the remaining budget (the amount not used by the ant) to improve the solution. In fact, some generated feasible solutions do not use the entire available budget.

This algorithm improves the initial solution by using this remaining budget to exchange some actual technologies by more reliable other technologies. A similar idea can be found in [10] where a neural network is presented to solve the job-shop scheduling problem, and where a similar procedure is used to improve the obtained solutions by eliminating the time segments during which all machines are idle.
The local search algorithm proceeds to change in turn each pair of chosen technologies by another pair. For each component, technologies are indexed in ascending order in accordance with their reliability. A solution $S = \{u, v, ...\}$ indicates that component 1 uses technology with index u, component 2 uses technology with index v, etc. Let consider for example a series system with 3 components and 6 available technologies for each component. Suppose that the obtained solution at a given cycle is $S = \{3, 2, 5\}$. The local search will evaluate the following solutions:

$$S=\{4, 1,5\},\ S = \{4, 2, 4\},\ S = \{2, 3, 5\},\ S = \{2, 2, 6\},\ S = \{3, 3, 4\},\ S = \{3, 1,6\}.$$

Among all these evaluated solutions, whenever an improvement feasible solution is detected, the new solution replaces the old one. It has been shown in [11] that the experimental results showed that the optimal or nearly optimal solutions could be obtained quickly. In the next section, Hybrid Ant Colony Optimization (HACO) will be used to solve the redundancy allocation problem. This HACO uses rather the extended great deluge algorithm as a local search within the proposed ant colony algorithm.

3. Redundancy allocation problem

3.1. Problem description

The redundancy allocation problem (RAP) is a well known combinatorial optimization problem where the design goal is achieved by discrete choices made from elements available on the market. The system consists of n components in series. For each component i ($i = 1, 2, ..., n$) there are various versions of elements, which are proposed by the suppliers on the market. Elements are characterized by their cost and weight according to their version. Each component i contains a number of elements connected in parallel. Different elements can be placed in parallel. A component i is functioning properly if at least k_i of its p_i elements are operational (k-out-of-n: G).
The series-parallel system is a logic-diagram representation for many design problems encountered in industrial systems. As it is pointed out in [18] and [31], electronics industry is an example where the RAP is very important. In fact, in this industry most systems require very high reliability and the products are usually assembled and designed using off-the-shelf elements (capacitors, transistors, microcontrollers, etc.) with known characteristics. Other examples where the above type of structure is becoming increasingly important include telecommunications systems and power systems. In all these systems, redundancy is indeed a necessity to reach the required levels of reliability and the RAP studied in this paper is therefore one of the major problems inherent to optimal design of reliable systems.
Assumptions

1. Elements and the system may experience only two possible states: good and failed.
2. The system weight and cost are linear combinations of element weight and cost.
3. The element attributes (reliability, cost and weight) are known and deterministic.
4. Failed elements do not damage the system, and are not repaired.

5. All redundancy is active: failure rates of elements when not in use are the same as when in use.
6. The supply of elements is unlimited.
7. Failures of individual elements are *s*-independent.

Notation

R_{sys}	overall reliability of the series-parallel system
R^*	optimal solution
C	cost constraint
W	weight constraint
n	number of components
i	index for components
a_i	number of available elements choices (i.e., versions) for component i
r_{ij}	reliability of element j available for component i
w_{ij}	weight of element j available for component i
c_{ij}	cost of element j available for component i
x_{ij}	number of element j used in component i
$\mathbf{x}_i$	$\left(x_{i1},\ x_{i2},\ \ldots,\ x_{ia_i}\right)$
p_i	total number of elements used in component i
p_{max}	maximum number of elements in parallel
k_i	minimum number of elements in parallel required for component i
$\mathbf{k}$	$(k_1, k_2, \ldots, k_n)$
$R_i(\mathbf{x}_i \mid k_i)$	reliability of component i, given k_i
$C_i(\mathbf{x}_i)$	total cost of component i
$W_i(\mathbf{x}_i)$	total cost of component i

The RAP is formulated to maximize system reliability given restrictions on system cost and weight. That is,

Maximize
$$R_{sys} = \prod_{i=1}^{n} R_i(\mathbf{x}_i \mid k_i) \tag{9}$$

Subject to
$$\sum_{i=1}^{n} C_i(\mathbf{x}_i) \le C, \tag{10}$$

$$\sum_{i=1}^{n} W_i(\mathbf{x}_i) \le W. \tag{11}$$

Constraints (10) and (11) represent respectively the budget and the weight constraints. If there is a pre-selected maximum number of elements which are allowed in parallel, the following constraint (12) is added:

$$k_i \le p_i \le p_{\max} \quad \forall\ i = 1,\ 2,\ \ldots,\ n. \tag{12}$$

3.2. Literature review

The RAP is NP-hard [17] and has previously been solved using many different optimization approaches and for different formulations as summarized in [39], and more recently in [32]. Optimization approaches to determine optimal or very good solutions for the RAP include dynamic programming, e.g. [12,25,35,41], mixed-integer and nonlinear programming, e.g.

[32], and integer programming, e.g. [14,27,28,34]. Nevertheless, these methods are limited to series-parallel structures where the elements used in parallel are identical. This constitutes a drawback since in practice many systems designs use different elements performing the same function, to reach high reliability level [31]. For example, as explained in [18], (airplanes use a primary electronic gyroscope and a secondary mechanical gyroscope working in *parallel*, and most new automobiles have a redundant (spare) tire with different size and weight characteristics forming a 4-out-of-5: G standby redundant system). Because of the above-mentioned drawback and of the exponential increase in search space with problem size, heuristics have become a popular alternative to exact methods. Meta-heuristics, in particular, offer flexibility and a practical way to solve large instances of the relaxed RAP where different elements can be placed in parallel.

Genetic algorithm (GA) is a well-known meta-heuristic used to solve combinatorial reliability optimization problems [18,33,37,42,43]. In addition to genetic algorithms, other heuristic or meta-heuristic approaches have also been efficiently used to deal with system reliability problems. A tabu search (TS) meta-heuristic [30] has been developed in [31] to efficiently solving the RAP, while the ant colony optimization meta-heuristic [20] is used in [4] to solve the problem.

In light of the aforementioned approaches, the method presented here gives a heuristic approach to solve the RAP. This method combines an *ant colony* optimization approach and a degraded *ceiling local* search technique. This approach is said to be *hybrid* and will be called *ACO/DC* (for Ant Colony Optimization and Degraded Ceiling).

The idea of employing a colony of cooperating agents to solve combinatorial optimization problems was recently proposed in [21]. The ant colony approach has been successfully applied to the classical traveling salesman problem [22,23], to the quadratic assignment problem [26], and to scheduling [13]. Ant colony shows very good results in each applied area. The ant colony has also been adapted with success to other combinatorial optimization problems such as the vehicle routing problem [15], telecommunication networks management [24], graph coloring [19], constraint satisfaction [38] and Hamiltonian graphs [44]. In [11], the authors used ant system to solve the optimal design problem of series system under budget constraints. The ant colony approach has also been applied for the RAP of multi-state systems in [36]. For the RAP in the binary state case, which is the focus of the present paper, the only existing work is that of [4].

3. 3. Hybrid solution approach: ACO/DC

As for the problem studied in section 2, to apply the AGO meta-heuristic to this problem, it is convenient to represent the problem by a graph $G = (\varsigma, \Lambda)$, where ς are the nodes and Λ is the set of edges. To represent our problem as such a graph, we introduce the following sets of nodes and edges [55]:

- *Three sets of nodes*:
  1. The first set of nodes (N_1) represents the components.
  2. The second set of nodes (N_2) represents, for each component, the numbers of elements which can be used in parallel. For example, if the maximum number of elements in parallel is three (p_{max} = 3), the set N_2 will be given by three nodes corresponding to one element, two parallel elements and three parallel elements.
  3. The third set (N_3) represents the versions of elements available for each component.

- *Two sets of edges*:
  1. The first set of edges is used to connect each component node in the set N_1 to the corresponding nodes in N_2.
  2. The second set of edges is used to connect the nodes in N_2 to the nodes in N_3 of their available versions.

Informally, our algorithm works as follows: m ants are initially positioned on a node representing a component. Each ant represents one possible structure of the entire system. This entire structure is defined by the vectors $\mathbf{x}_i$ (i = 1, ..., n). Each ant builds a feasible solution (called a tour) to our problem by repeatedly applying stochastic greedy rules (i.e., the *state transition rules*). Once all ants have terminated their tour, the amount of pheromone on edges is modified by applying the *global updating rule*. Ants are guided, in building their tours, by both heuristic information (they prefer to choose "less expansive" edges), and by pheromone information. Once an ant has built a structure, the obtained solution is improved by using a local search algorithm. This step is performed only in the following cases:

- the obtained solution by the ant is feasible and,
- the quality of the solution is "good". The term "good" means here that the reliability R_{sys} of the structure should be either better than the best solution R^*, i.e., $R_{sys} \geq R^*$,or close to this best solution R^*, i.e., $0 \leq \frac{R^* - R_{sys}}{R^*} \leq +l$ where l represents the solution quality level.

Ants can be guided, in building their tours, by pheromone information and heuristic information. Naturally, an edge with a high amount of pheromone is a very desirable choice. The pheromone updating rules are designed so that they tend to give more pheromone to edges which should be visited by ants.

In the following we discuss the state transition rules and the global updating rules.

State transition rules

In the AGO algorithm, each ant builds a solution (called a tour) to our problem by repeatedly applying two different *state transition rules*. At each step of the construction process, ants use: (1) pheromone trails (denoted by τ_{ij}) to select the number of elements connected in parallel and the versions of elements; (2) and a problem- specific heuristic information (denoted by η_{ij}) related to the versions of elements.

An ant positioned on node i (i.e. component i) chooses the total number p_i of elements to be connected in parallel. This choice is done by applying the rule given by:

$$P_{ip_i} = \begin{cases} \dfrac{\left(\tau_{ip_i}\right)^{\alpha_1}}{\sum_{k=1}^{p_{\max}} \left(\tau_{ik}\right)^{\alpha_1}} & if \quad p_i \in \{1,\ 2,\ ..,\ p_{\max}\} \\ 0 & otherwise \end{cases} \tag{13}$$

where α_1 is a parameter that controls the relative weight of the pheromone (τ_{ip_i}). We favour the choice of edges which are weighted with greater amount of pheromone.

When an ant is positioned on node p_i representing the selected number of elements connected in parallel in component i, it has to choose these p_i versions of elements. This choice is done by applying the rule given by:

$$P_{p_i j} = \begin{cases} \dfrac{\left(\tau_{p_i j}\right)^{\alpha_2} \left(\eta_{p_i j}\right)^{\beta}}{\sum_{k=1}^{a_i} \left(\tau_{p_i k}\right)^{\alpha_2} \left(\eta_{p_i k}\right)^{\beta}} & if \quad j \in \{1,\ 2,\ ..,\ a_i\} \\ 0 \quad otherwise \end{cases} \tag{14}$$

where α_2 and β are respectively parameters that control the relative weight of the pheromone ($\tau_{p_i j}$) and the local heuristic ($\eta_{p_i j}$).

We tested different heuristic information and the most efficient was $\eta_{p_i j} = (r_{ij}) / (w_{ij}^3)$ where r_{ij} and w_{ij} represent respectively the associated reliability and weight of version j for component i. In equation (14) we multiply the pheromone on edges by the corresponding heuristic information. In this way we favour the choice of edges which are weighted with smaller weight and greater reliability and which have a greater amount of pheromone.

Global updating rule

During the construction process, no guarantee is given that an ant will construct a feasible solution which obeys the constraints (11) and (12). The unfeasibility of solutions is treated in the pheromone update: the amount of pheromone deposited by an ant is set to a high value if the generated solution is feasible and to a low value if it is infeasible. These values are dependent of the solution quality. Infeasibilities can then be handled by assigning penalties proportional to the amount of cost and weight violations. Thus, the trail intensity is updated as follows:

$$\tau_{ij}(new) = \rho \tau_{ij}(old) + \Delta \tau_{ij} \tag{15}$$

ρ is a coefficient such that (1 - ρ) represents the evaporation of trail and $\Delta \tau_{ij}$ is :

$$\Delta \tau_{ij} = \sum_{k=1}^{m} \Delta \tau_{ij}^{k} \tag{16}$$

where m is the number of ants. Furthermore, $\Delta \tau_{ij}^{k}$ is given by:

$$\Delta \tau_{ij}^{k} = \begin{cases} Q \ penalty_k \ R_{sys}^{k} & \text{if the edge } (i, j) \text{ is visited by the } k^{\text{th}} \text{ ant} \\ 0 & \text{otherwise} \end{cases} \tag{17}$$

where Q is a positive number, R_{sys}^{k} is the system reliability for ant k, and $penalty_k$ is defined as follows:

$$penalty_k = \begin{cases} \left(\dfrac{C}{TC_k}\right)^{a} & if\ (C < TC_k) \\ \left(\dfrac{W}{TW_k}\right)^{b} & if\ (W < TW_k) \\ \left(\dfrac{C}{TC_k}\right)^{a} \left(\dfrac{W}{TW_k}\right)^{b} & if\ (C < TC_k \ and \ W < TW_k) \end{cases} \tag{18}$$

and TW_k are respectively the total cost and the total weight obtained by ant k. Parameters α and β represent the relative importance of penalties.

The degraded ceiling local search meta-heuristic

The principle of the degraded ceiling meta-heuristic

The performance of algorithms based on the AGO meta-heuristic can be greatly enhanced when coupled to local improvement procedures. A degraded ceiling (DC) based algorithm is included in our AGO approach to improve the solutions obtained by the ants.

The degraded ceiling is a local search meta-heuristic recently introduced in [6a] and [6b]. Like other local search methods, the degraded ceiling iteratively repeats the replacement of a current solution s by a new one s^*, until some stopping condition has been satisfied. The new solution is selected from a neighbourhood $N(s)$. The mechanism of accepting or rejecting the candidate solution from the neighbourhood is different of other methods. In degraded ceiling approach, the algorithm accepts every solution whose objective function is more or equal (for the maximization problems) to the upper limit L, which is monotonically increased during the search by ΔL.

The initial value of ceiling (L) is equal to the initial cost function $f(s)$ and only one input parameter ΔL has to be specified. In [16], the authors applied successfully the degraded ceiling on exam timetabling problem and demonstrated that it outperformed well-known best results found by others meta-heuristics, such as simulated annealing and tabu search. They showed two main properties of the degraded ceiling algorithm:

- The search follows the degrading of the ceiling. Fluctuations are visible only at the beginning, but later, all intermediate solutions lie close to a linear line.
- When a current solution reaches the value where any further improvement is impossible, the search rapidly converges. The search procedure can then be terminated at this moment.

The degraded ceiling algorithm is an extension of the "great deluge" method which was introduced as an alternative to simulated annealing. Degraded ceiling, simulated annealing and "great deluge" algorithms share the characteristic that they may both accept worse candidate solutions than the current one. The difference is in the acceptance criterion of worse solutions. The simulated annealing method accepts configurations which deteriorate the objective function only with a certain probability. The degraded ceiling algorithm incorporates both the worse solution acceptance (of the "great deluge" algorithm) if the solution fitness is less than or equal to some given upper limit L, i.e. ($f(s^*) \geq L$), and the well-known hill climbing rule ($f(s^*) \geq f(s)$).

Like simulated annealing, the degraded ceiling algorithm may accept worse candidate solutions during its run. The introduction of the dynamic parameter has an important effect on the search. As explained in [16], the decreasing of L may be seen as a control process, which drives the search towards a desirable solution. Note finally that degraded ceiling algorithm has the advantage to require only one parameter (ΔL) to be tuned.

3.4. Test problems and results

The test problems, used to evaluate the performance of the ACO/DC methodology for the RAP, were originally proposed by Fyffe *et al*. in [25] and modified by Nakagawa and Miyazaki in [35]. Fyffe *et al*. [25] specified constraint limits of 130 units of system cost, 170 units of system weight and $k_i = 1$ (i.e., l-out-of-n:G). Nakagawa and Miyazaki [35] developed 33 variations of the original problem, where the cost constraint C is set to 130 units and the

weight constraint W varies from 159 units to 191 units. The system is designed with 14 components. For each component, there are three or four element choices [55].

Earlier optimization approaches to the problem (e.g., [25] and [35]), required that only identical elements be placed in redundancy. Such approaches determined optimal solutions through dynamic programming and IP models, but only a restricted set of solutions was considered due to computational or formulation limitations of exact solution methods. Nevertheless, for the ACO/DC approach, as in [81], [31] and [4], different types are allowed to reside in parallel. In [18], Coit and Smith first solved the RAP with a genetic algorithm without restricting the search space. More recently, Kulturel-Konak *et al*. solved this problem in [31] with a tabu search algorithm, while Liang and Smith [31] used an ant colony optimization approach improved by a local search. Because the heuristic benchmarks for the RAP where elements mixing is allowed are the methods in [18], [31] and [4], there are chosen for comparison. Our approach will be compared also with [35] and [29]. By comparing the proposed ACO/DC methodology to all the above-mentioned papers (e.g., [18], [31], [4], [35] and [29]), we compare it to the best-known solutions found in literature at the best of our knowledge.

In meta-heuristics such as AGO, simulated annealing and degraded ceiling, it is necessary to tune a number of parameters to have good performance. The user-specified parameters of the ACO/DC algorithm were varied to establish the values most beneficial to the optimization process. Following the tuning procedure used in [21-23], we tested various values for each parameter, while keeping the others constant. Based on these initial experiments the values found to be most appropriate are:

$$\alpha_1 = 0.1,\ \alpha_2 = 0.5,\ \beta = 1,\ Q = 0.01,\ \rho = 0.9,\ a = 1,\ b = 10,\ \tau_0 = 1,\ \Delta L = 0.0001 \text{ and } l = 0.01.$$

These parameters values are used for all test problems. 50 ants are used in each iteration. When combined to the degraded ceiling algorithm, AGO converges quickly to optimal or near optimal solutions. Note that the degraded ceiling is called only if the obtained solutions are very good. For the considered problem instances, the maximum number of iterations needed does not exceed 300 iterations.

Comparing the results obtained by our approach with those of previous works [18,29,31,4,35], it has been shown in [55] that:

1. The solutions found by our algorithm are all better than those of Hsieh [19].
2. In 31 of the 33 test cases, the solutions found by our algorithm are better than those of Nakagawa and Miyazaki [27] while the rest (i.e., 2 cases) are as good as those they found.
3. Cases 22 to 29 and 31 to 32 are as good as those found by the genetic algorithm of Coit and Smith [8] while the rest (i.e., 24 instances) are all better than those they found.
4. In 6 of the 33 test cases, the ACO/DC outperformed the tabu search algorithm of Kulturel-Konak *et al*. [21] while it was very close but at a lower reliability in only 2 instances.
5. In 9 of the 33 test cases, the solutions found by our algorithm are better than those of Liang and Smith [24] while the rest are as good as those they found.

Both the degraded ceiling and the ant colony algorithms are meta-heuristics. Our contribution is based on the ACO/DC hybridization and very good results are obtained. The RAP is one of the most difficult combinatorial optimization problems inherent to optimal design of reliable systems. We believe and we show that two efficient meta-

heuristics have to cooperate in order to solve efficiently this problem, namely the AGO and the DC meta-heuristics.
The study conducted in this section shows again that hybridization of meta-heuristics is a very promising methodology for NP-hard reliability design problems.

4. Selecting machines and buffers in series-parallel production lines

4.1. Optimal design problem

Consider the series-parallel production line. Buffers are inserted to limit the propagation of disruptions, and this increases the average production rate of the line. This line consists of n components and n-l buffers. Each component of type i (i = 1,2,..., n) can contain several identical machines connected in parallel. For each component i, there are a number of machine versions available in the market.
In order to formulate the problem in mathematical expression, the following notations are introduced first:

h_i version number of machine i
H_i maximum h_i available
h $\{ h_i \}$, $h_i \in \{1,2,..,H_i\}$
r_i number of elements connected in parallel in i
R_i maximum r_i allowed
$C(h_i)$ cost of each machine of version h_i
$P(h_i)$ isolated production rate of machine with version h_i
$T(h_i)$ processing time of machine with version h_i
$\lambda(h_i)$ failure rate of each machine with version h_i
$\mu(h_i)$ repair rate of each machine with version h_i
$u(h_i)$ speed of the machine's version h_i

We assume that a buffer is also chosen from a list of available buffers. The buffers of different versions f differ by their size N^b_f and cost C^b_f. The total number of different buffer versions available for m^{th} component is denoted by F_m. The vector $\mathbf{f} = \{f_m\}$, where $0 \le f_m \le F_m$, defines versions of buffers chosen for each component. The entire production line structure is defined by the vectors **h**, **r** and $\mathbf{f} = \{f_m\}$.
For the given **h**, **r** and **f**, the total cost of the production line can be calculated as:

$$C_T = \sum_{m=1}^{n-1} C^B_{f(m)} + \sum_{i=1}^{n} r_i . C(h_i) \tag{19}$$

The optimal design problem of production system can be formulated as follows: find the system configuration **f**, **h** and **r** that maximizes the total production rate P_T such that the total cost C_T is less or equal to a specified value C^*. That is,

Maximize $P_T(\mathbf{f}, \mathbf{h}, \mathbf{r})$ (20)

Subject $C_T(\mathbf{f}, \mathbf{h}, \mathbf{r}) \le C^*$ (21)

The input of this problem is C^* and the outputs are the optimal production rate $P_{T\text{Max}}$ and the corresponding configuration determined by the vectors **f**, **h**, **r**. The resulting maximum value of P_T is written $P_{T\text{Max}}(C^*)$.

4.2. Literature review

There is a substantial literature on the analysis of production lines with buffers [45]. This literature is mainly concerned with the prediction of performance. Much of it is aimed at evaluating the average production rate (throughput) of a system with specified machines and buffers. In [46], the authors present a set of algorithms that select the minimal buffer space in a flow line to achieve a specified production rate. The algorithms are based on analytical approximations of the Buzacott model of a production line [47,48]. For a recent review of the literature on production line optimization, the reader is referred to [46]. The goal of the existing works is to choose buffers sizes for a production line. They all assume that the number of machines is specified, and the only parameters to find are buffers sizes. The proposed approach to optimal design aims at *selecting both buffers and machines*; it gives also the number of redundant machines used in parallel.

To deal with the optimal design problem considered in this work, it is mandatory to develop a method for throughput evaluation of series-parallel manufacturing production lines. This method has to take into account two characteristics:

(i) Components may consist of banks of parallel machines. Concerning this first characteristic, we attempt to represent each stage by an equivalent single component.

(ii) The processing rate may differ from component to component. To deal with this second characteristic, we use a continuous (or fluid) material model type which produces very good results. This consists of two main steps. First, the non homogeneous line is transformed into an approximately equivalent homogeneous one. In a second step, the resulting homogeneous line is analysed by using the well-known decomposition method for homogeneous lines [49].

The effect of the used simplifications for estimating throughput is examined by comparing the results provided by our approximate technique to simulation results on many examples. This comparison shows that the proposed approximate technique is very accurate.

As the formulated problem is a complicated combinatorial optimization one, an exhaustive examination of all possible solutions is not realistic, considering reasonable time limitations. Because of this, we develop two heuristics to solve the problem. The first heuristic is inspired from the *ant colony system* meta-heuristic: a recent kind of biologically inspired algorithms [48,49]. The second proposed heuristic is based on the *simulated annealing* meta-heuristic [50].

4.3. Throughput evaluation of series-parallel production lines

4.3.1. Summary of the method

The proposed method approximates each component (i.e. each set of parallel machines) of the original production line as a single unreliable machine. The system is then reduced to a single machine per component production line of the type represented in figure 2. The equivalent machines may have different processing rates. To determine the steady state behaviour of this *non-homogeneous* production line, it is first transformed into an approximately equivalent homogeneous line. Then, the well-known Dallery-David-Xie algorithm (DDX) is used to solve the decomposition equations of the resulting (homogenous) line [51].

4.3.2. Replacing each component by an equivalent machine

The decomposition techniques developed in the literature are efficient in estimating performance characteristics of series production lines. In these techniques it is necessary for

each component to be described by three parameters: *failure rate*, *repair rate* and *processing rate*. By limiting the description of the equivalent machine to these three parameters, our analysis of the new system is reduced in complexity to that of the existing decomposition techniques. Furthermore, the state space of a series-parallel line grows large with the number of parallel machines in the components. Replacing each set of parallel machines by one equivalent machine leads advantageously to a reduction of the state space.

Let denote by λ_{ij}, μ_{ij} and P_{ij}, respectively, the failure rate, the repair rate and the processing rate of a machine M_{ij}, and by λ_i, μ_i and P_i, respectively, the failure rate, the repair rate and the processing rate of a machine M_i. To calculate the three unknown quantities λ_i, μ_i and P_i, we have to formulate three equations. Assuming that machines within the set of parallel machines are fairly similar, the following approximation is proposed in [52] :

$$\lambda_i = \sum_{j=1}^{J_i} \lambda_{ij} \qquad i = 1, 2, \ldots, n \tag{22}$$

$$\mu_i = \sum_{j=1}^{J_i} \mu_{ij} \qquad i = 1, 2, \ldots, n \tag{23}$$

$$P_i = \sum_{j=1}^{J_i} P_{ij} \qquad i = 1, 2, \ldots, n \tag{24}$$

It is shown in [52] that it is a good approximation. However, when buffers are small, this heuristic is inaccurate. In the present work, we assume that the available buffers are large enough. Thus, each set of parallel machines is approximated as an equivalent single machine by using equations (22), (23) and (24). This leads to a non-homogenous line. Therefore, a homogenisation technique is required, as explained in the next subsection.

4.3.3. Homogenisation technique

It is known that in the case of non-homogenous lines (*i.e.* production lines in which the machines do not have the same processing time), two approaches can be used. The first approach is based on an extension to the case of homogenous lines of the decomposition technique developed in [49]. The second approach relies on the modification of the non-homogeneous line into an approximately equivalent homogeneous line by means of *homogenisation techniques* [53]. The analysis of the obtained homogeneous line is therefore based on the use of the decomposition method for homogeneous line. In this way, it is possible to rely on the DDX algorithm which has been proven to be very fast and reliable. In [53], the authors showed that the homogenization method of [54], referred to as the completion time approximation, provides fairly accurate results. In this method, each machine M_i of the original non-homogeneous line is replaced by an approximately equivalent machine M_i^e, such that its completion time distribution is close to that of the original machine. The processing time of machine M_i^e is set to the processing time of the fastest machine in the original line: $T^e = \min(T_1, T_2, \ldots, T_k)$. Since the processing time of M_i^e is given (equal to T^e) there are two parameters per machine that must be determined, namely the failure and repair rates. Let λ_i^e and μ_i^e be the failure and repair rates of machine M_i^e. The principle of the method developed in [53] is to determine λ_i^e and μ_i^e in such a way that the distribution of completion times of machine M_i^e has the same first and second moments as

those of the distribution of completion times of machine M_i. The values of λ_i^e and μ_i^e are given in [53] by:

$$\lambda_i^e = \left[\frac{T_i}{T^e}\left(1+\frac{\lambda_i}{\mu_i}\right)-1\right]^2 \frac{T^e}{T_i}\frac{\mu_i^2}{\lambda_i} \text{ and } \mu_i^e = \left[\frac{T_i}{T^e}\left(1+\frac{\lambda_i}{\mu_i}\right)-1\right]\frac{T^e}{T_i}\frac{\mu_i^2}{\lambda_i}.$$

4.3.4. Decomposition equations and DDX algorithm

As said before, we denote by λ_i^e, μ_i^e and T_i^e respectively, the failure rate, the repair rate and the processing time of the machine M_i^e in the equivalent homogenous line. In [51], the authors developed decomposition equations for homogenous lines and propose an efficient algorithm (DDX) to solve these equations.

Production line decomposition methods typically work as follows. An original line is divided into *k-1* lines with only two machines. The method requires the derivation of a set of equations that link the decomposed systems together. Such methods are efficient because systems with two machines can be rapidly analyzed. In general, systems may be represented by discrete or continuous flow models. In both, the processing time is deterministic. The discrete material model has the advantage of better representing the discrete nature of typical factories, but it is limited to systems with equal processing times. The continuous (or fluid) model is better suited in our case because it can be used for systems where the machines have different processing rates. The fluid modelling approach is an approximation which consists in using continuous variables to characterize the flow of parts. Therefore, the quantity of material in each buffer B_i at any time t is a real number taking its value in the interval $[0, N_i]$.

The DDX algorithm [51] is the quickest and most reliably convergent algorithm for solving decomposition-type equations. In our optimal design problem, the DDX algorithm can be used to solve the decomposition equations for each configuration. Let recall that in our analytical method the DDX algorithm is applied after approximating each set of parallel machines as a single machine and transforming the resulting non-homogenous production line into an approximate equivalent homogenous line. For more details about DDX algorithm, the reader is referred to [51].

4.4. The hybrid ant colony optimization (HACO) and the simulated annealing

4.4.1. Applying ACS to select machines and buffers: the general algorithm

Following [21], with respect to the problem of selecting machines and buffers in a series-parallel line, each ant is an agent that leaves a pheromone trail, called a trace, on the edges of a graph representing the problem. To represent our problem as such a graph, we introduce the following sets of nodes and edges [56] :

- *Three sets of nodes*:

1. The first set of nodes (N_1) represents the components and the buffers.
2. The second set (N_2) represents the versions of elements available for each component and buffer.
3. The third set of nodes (N_3) represents, for each component, the numbers of elements which can be used in parallel. For example, if the maximum number of elements in parallel is two, the set N_3 will be given by two nodes corresponding to one element and two parallel elements.

- *Two sets of edges*:

1. The first set of edges is used to connect each node in the set N_1 to the corresponding nodes in N_2.
2. The second set of edges is used to connect some nodes in N_2 to the nodes in N_3.

Informally, our algorithm works as follows: m ants are initially positioned on a node representing a component. Each ant represents one possible structure of the entire system. This entire production line structure is defined by the vectors **f**, **h** and **r**. Each ant builds a feasible solution (called a tour) to our problem by repeatedly applying *three* different stochastic greedy rules (i.e., the *state transition rules*). While constructing its solution, an ant also modifies the amount of pheromone on the visited edges by applying the *local updating rule*. Once all ants have terminated their tour, the amount of pheromone on edges is modified again (by applying the *global updating rule*). Ants are guided, in building their tours, by both heuristic information (they prefer to choose "less expensive and more efficient edges"), and by pheromone information.

Note that when an ant builds a solution, it can be feasible or unfeasible. When the obtained solution is unfeasible, it is automatically rejected and it is not taken into account in the comparison with the other feasible solutions obtained by the other ants. It should be noted also that the global update of the pheromone is done only for the best obtained feasible solution.

In the following we discuss the state transition rules, the global updating rule, and the local updating rule.

State transition rules

In the above algorithm, at each step of the construction process, ants use: (1) pheromone trails (denoted by τ_{ij}) to select the versions of machines and buffers and the number of machines connected in parallel; (2) a problem-specific heuristic information (denoted by η_{ij}). The value of η_{ij} depends of the nature of the node (i.e. machine's version or buffer's version). Note that the choice of the number of machines to be connected in parallel is not function of the heuristic information η_{ij}.

An ant positioned on node i (representing a machine or a buffer) chooses the version j ($j= h_i$ if i is a machine and $j =f_i$ if i is a buffer) according to following:

$$j = \begin{cases} \arg\max_{k \in \Gamma_i} \left\{ (\tau_{ik})^{\alpha} (\eta_{ik})^{\beta} \right\} & \text{if} \quad q \le q_o \\ J & \text{otherwise} \end{cases} \tag{25}$$

where Γ_i is the set of nodes representing the available versions for node i (Γ_i={ 1,...,H_i} if i is a machine or Γ_i={1,... ,F_i} if i is a buffer).

And J is a random variable selected according to the probability distribution given by:

$$p_{ij} = \begin{cases} \dfrac{(\tau_{ij})^{\alpha} (\eta_{ij})^{\beta}}{\sum_{k \in \Gamma_i} (\tau_{ik})^{\alpha} (\eta_{ik})^{\beta}} & \text{if} \quad j \in \Gamma_i \\ 0 & \text{otherwise} \end{cases} \tag{26}$$

In the above equations (25) and (26), α and β are parameters that control the relative weight of the pheromone (τ_{ij}) and the local heuristic (η_{ij}), respectively. The value of β depends on

the type of a given node. The variable q is a random number uniformly distributed in [0, 1]; and q_0 is a parameter ($0 \leq q_0 \leq 1$) which determines the relative importance of exploitation versus exploration $\eta_{ij} = \frac{P(h_i)}{C(h_i)}$ if i represents a machine and $\eta_{ij} = \frac{N(f_i)}{C(f_i)}$ represents a buffer.

Similarly, when an ant is positioned on node i representing a version of a machine, it has to select a number j of machines to be connected in parallel. In this case, the used rule is similar to (7) and (8) except for the heuristic information which is set to 1 and $\Gamma_i = \{1,..., R_i\}$.

Global updating rule

Once all ants have built a complete solution, pheromone trails are updated. Only the globally best ant (i.e., the ant which constructed the best design solution from the beginning of the trial) is allowed to deposit pheromone. A quantity of pheromone $\Delta\tau_{ij}$ is deposited on each edge that the best ant has used, where the indices i and j refer to the edges visited by the best ant. The quantity $\Delta\tau_{ij}$ is given by the total production rate P_{Tbest} of the design feasible solution constructed by the best ant. Therefore, the global updating rule is:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \rho\Delta\tau_{ij} \tag{27}$$

where $0 < \rho < 1$ is the pheromone decay parameter representing the evaporation of trail.

Global updating is intended to allocate a greater amount of pheromone to greater design solution. Equation (27) dictates that only those edges belonging to the globally best solution will receive reinforcement.

Local updating rule

While building a solution of the problem, ants visit edges on the graph G, and change their pheromone level by applying the following local updating rule:

$$\tau_{ij} \leftarrow (1-\rho)\tau_{ij} + \rho\tau_o \tag{28}$$

where τ_o is the initial value of trail intensities.

The application of the local updating rule, while edges are visited by ants, has the effect of lowering the pheromone on visited edges. This favours the exploration of edges not yet visited, since the visited edges will be chosen with a lower probability by the other ants in the remaining steps of an iteration of the algorithm.

Improving constructed solutions

As said before, it is well known that the performance of ACS algorithms can be greatly improved when coupled to local search algorithms [2]. Following this idea again, an improvement procedure is included in our ACS algorithm, once all ants have terminated their tour and before applying the global updating rule.

This procedure consists of two steps:

1. The remaining budget (the amount not used by the ant) of the obtained structure is first used to improve the solution. In fact, some generated feasible solutions do not use the entire available budget. The procedure improves the initial solution by using this remaining budget to increase the number of machines connected in parallel.
2. In this step, two types of evaluation are performed depending of the nature of the component *(i.e.* machine or buffer). For each pair of components representing the machines, the number of machines is changed by adding one for the first component and subtracting one for the second component. In the case of buffers, the algorithm proceeds to change in turn each pair of chosen versions by another pair.

The above steps are illustrated in the following example. Let us consider a series-parallel line with 3 machines (5 available versions for each machine) and 2 buffers (6 available versions for each buffer). Suppose that the solution at a given cycle is **f** = {3, 2}, **h** = {2, 1, 1} and **r** = {2, 3, 3}. The improvement procedure will evaluate the structures with the following numbers of parallel machines:

r ={1,4,3}, **r** ={1,3,4}, **r** ={2,2,4}, **r** ={3,2,3}, **r** ={3,3,2} and **r** ={2,4,2},

and the following versions of buffers:

f = {2,3} and **f** = {4,1}.

Note finally that when this improvement procedure is used, only the neighbouring feasible solutions are evaluated and compared with the current solution.

4.4.2. Simulated annealing for the optimal design of series-parallel lines

The simulated annealing (SA) exploits the analogy between the way in which a metal cools and freezes into a minimum crystalline energy and the search for a minimum in a more general energy. The connection between this algorithm and mathematical minimization was first noted by [57], but it was Kirckpatrick *et al*. in 1983 [50] who proposed that it forms the basis of optimization technique for combinatorial problems.

The simulated annealing technique is an optimization method suitable for combinatorial minimization problems. A new solution is generated and compared against the current solution. The new solution is accepted as the current solution if the difference in quality does not exceed a dynamically selected threshold. The solutions corresponding to larger increases in cost have a small probability of being accepted. A parameter that regulates the threshold is called the temperature and the function that determines the values for the temperature over time is called the cooling scheduling. The temperature decreases over time to decrease the probability of non improving moves.

Initial feasible solution

The initial feasible solution can be generated in many ways. We tried two generation methods. The first one generates a feasible initial solution by taking the least expensive solution (i.e. only one machine in each component and version 1 for all buffers and machines). The second way starts with the least expensive solution and tries to improve it by an iterative improvement procedure. The experimental tests show that the first method is better.

Neighbouring solution

There are many ways to define neighbourhood for this problem. On the one hand, two types of neighbourhood structures have been tested. Regarding the number of machines in parallel for example, the first type was adding or subtracting one machine. The second one consisted in choosing a random number of machines in parallel. This kind of neighbourhood move has been proposed also in [58] when solving the buffer allocation problem. The carried out experiments showed that the second way is slightly more effective. On the other hand, the versions of the machines are indexed in ascending order of the production rate $P(h_i)$.

Our adopted neighbouring structure can be summarized by the following steps:

Step 1.

Randomly select a component COMP representing either a machine or a buffer.

Step 2.

If COMP = Machine, randomly select one of these two actions:

Action 1: Change the number of machines in parallel by choosing a random number less than k_{max} (k_{max} is the maximum number of machines allowed to be connected in parallel).

Action 2: Change the version of machine VERSION(Machine) by VERSION(Machine)+1 or VERSION(Machine)-1.

If COMP = Buffer, change its version VERSION(Buffer) by VERSION(Buffer)+1 or VERSION(Buffer)-1.

When a neighbour solution is randomly selected, it can be either feasible or unfeasible. If the solution is unfeasible, it is automatically rejected without using the criterion of acceptance and the algorithm passes to the next iteration.

Before introducing the numerical results, it should be noted that it would be straightforward to iterate the improvement procedure until no further improvements are found, i.e. to turn it into a local hill-climber. The coupling of a local search procedure such as the hill-climbing with the ACES may give a good idea on the quality of the obtained solutions. However, this will increase considerably the total computation time. Because the calculation of the objective function depends greatly on the convergence of the DDX algorithm whose time is not negligible, we proposed a local search procedure which does not require much evaluations of the objective function as the hill-climbing.

Numerical results

To prove the efficiency of our algorithm when combined with the local search, we proposed four examples of production line with respectively 4, 10, 15 and 20 components. Tables A.1-A.7 (in Appendix) show the corresponding data. The versions are indexed in ascending order of the production rate $P(h_i)$. The available budgets are respectively 160$, 300$, 450$ and 750$ and the maximum number of machines allowed to be connected in parallel is 3 for the first example and 4 for the other examples. The search space size is respectively larger than 5.5×10^5, 2.684×10^{18}, 8.796×10^{27} and 2.88×10^{37}. All the algorithms were implemented using MATLAB on a PC with 1.8 GHz processor.

	Example 1 ($n = 4$)	Example 2 ($n = 10$)	Example 3 ($n = 15$)	Example 4 ($n = 20$)
ACO$^{(*)}$	α=0.1, β_1=0.5, β_2=1, ρ=0.01, τ_o=1/25, q_0=0.75.	α=0.1, β_1=0.1, β_2=0.3, ρ=0.01, τ_o=1/25, q_0=0.75.	α=1, β_1=1.5, β_2=1, ρ=0.05, τ_o=1/15, q_0=0.80.	α=1, β_1=1.5, β_2=1, ρ=0.05, τ_o=1/15, q_0=0.80.
ACO$^{(*)}$ with improving procedure	α=0.1, β_1=0.1, β_2=1, ρ=0.05, τ_o=1/25, q_0=0.80.	α=0.5, β_1=1, β_2=1, ρ=0.05, τ_o=1/50, q_0=0.80.	α=0.5, β_1=1, β_2=1, ρ=0.05, τ_o=1/50, q_0=0.75.	α=1, β_1=0.3, β_2=0.5, ρ=0.05, τ_o=1/35, q_0=0.75.
Simulated annealing	c=0.98, T=5, length inner loop=40, $T_{min}=5.10^{-5}$, $V_{max}=40$	c=0.99, T=5, inner loop=100n, $T_{min}=5.10^{-5}$, $V_{max}=35n$	c=0.95, T=5, inner loop=100n, $T_{min}=5.10^{-5}$, $V_{max}=35n$	c=0.975, T=5, inner loop=100n, $T_{min}=5.10^{-5}$, $V_{max}=35n$

Table 1. Parameters data for three typical examples taken from [52]
(*) $\beta=\beta_1$ when the node i of equations (25) and (26) is a machine and (*) $\beta=\beta_2$ otherwise

We implemented the simulated annealing and ACS algorithm, and we ran simulations to set the parameters. For the ACS algorithm, the parameters considered are those that effect directly or indirectly the computation of the probability in formulas (25) and (26) (*i.e.* α, β, ρ, τ_o and q_0). We tested several values for each parameter, while all the others were held constant (over ten simulations for each setting in order to achieve some statistical

information about the average evolution). Based on these initial experiments the values found to be most appropriate are determined. Furthermore, in all our experiments, the number of ants is set to 5. Note that the ACS is not very sensitive to changes in these values, and tested well for quite a range of them. The parameters considered for the simulated annealing are the initial temperature T, length of the inner loop, the final temperature T_{min}, the maximum of success solution V_{max} and the cooling rate c. Initially, the temperature T is set to a sufficiently high value to accept all solutions during the initial phase of the simulated annealing. The cooling rate c should be generally greater than 0.7. Table 1 shows the values of all the parameters considered in the three algorithms.

Each algorithm was tested by performing ten trials. Figures 2 to 5 show the highest throughput versus the number of evaluations. By 6000, 200.000, 250.000 and 400.000 evaluations of throughput, respectively, for example 1, 2, 3 and 4, the highest throughput has been leveled out. These numbers of evaluations are used to assess the performance of the algorithms.

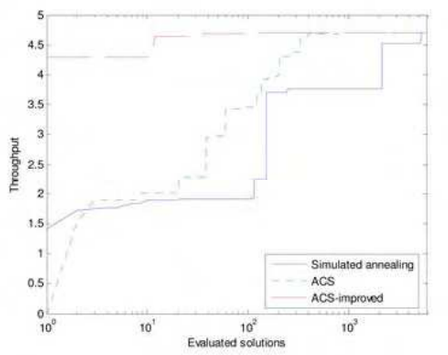


Figure 2. Convergence results for example 1

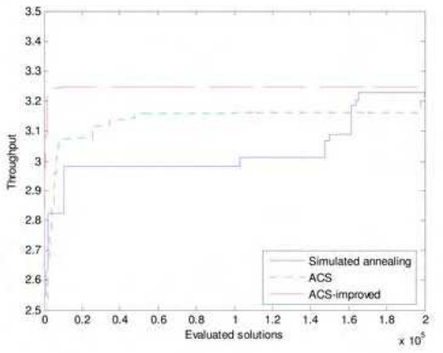


Figure 3. Convergence results for example 2

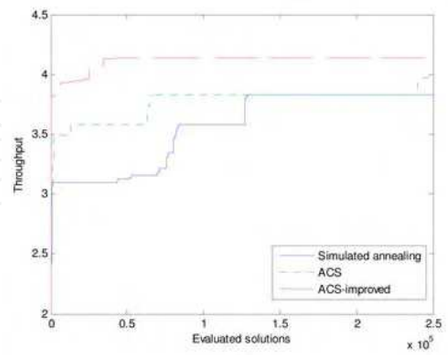


Figure 4. Convergence results for example 3

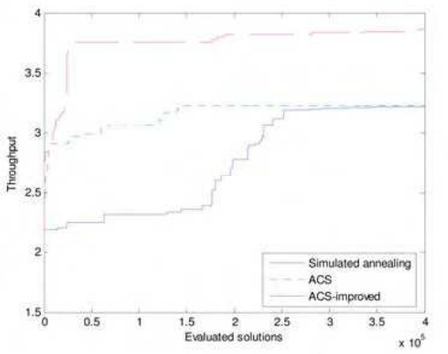


Figure 5. Convergence results for example 4

The convergence curves in figures 2 to 5 show that the ACS algorithm performs better when coupled with the improvement procedure. Generally, the convergence is faster and the quality is better than the other algorithms. The ACS algorithm when coupled with the local improvement procedure starts with a good solution, because the initial solutions built by the ants are improved by the procedure at the *first iteration* and before being reported in the graph. It is important to note that *all* the evaluated solutions are taken into account including those generated by the local improvement procedure. The results obtained by simulated annealing and ACS without the improvement procedure are fairly similar in the 4 examples.

The results obtained after 10 trials are given in tables 2 and 3. The solutions obtained by the ACS when coupled with the improvement procedure are the best obtained solutions. The application of the improvement procedure with the ACS improves the quality of solutions and the time required to produce near optimal solutions. Table 6 shows that the best results obtained by the simulated annealing and ACS (without the improvement procedure) are almost similar. However, we remark that:

(i) The mean values of the results obtained by the simulated annealing are clearly better than those obtained by the ACS algorithm.

(ii) The execution times of simulated annealing and ACS when coupled with the improvement procedure are lower than the execution time of ACS alone. For instance, in example 4 the mean execution time is 3960 seconds for ACS alone and it is about 1172 and 2030 seconds for SA and ACS coupled with the improvement procedure, respectively.

	P_{Tmax}	C_T(\$)	**H**	**r**	**f**
Example 1	4.7074	160	(1,3,1,1)	(3,3,3,3)	(2,1,2)
Example 2	3.2467	250	(1,1,1,1,5,5,1,2,1,2)	(2,2,2,3,1,1,2,1,1,1)	(1,1,1,1,3,1,1,1,1)
Example 3	4.4406	450	(1,1,1,1,1,1,1,1,1,1,1,2,1,1)	(3,2,2,4,2,2,2,2,2,2,2,4,2,2)	(1,1,2,1,1,1,1,1,1,1,1,2,1,1)
Example 4	3.8991	750	(2,4,3,1,5,4,2,4,1,2,1,2,1,2,1,1,2,3,4,2)	(3,2,2,4,2,2,2,2,4,2,3,2,3,2,4,3,2,2,2,2)	(1,2,2,1,1,3,1,2,2,1,1,1,1,1,1,2,1,1,1)

Table 2. Results for examples 1, 2, 3 and 4

	Simulated annealing					**ACS**					**ACS with improving procedure**				
	Min	**Mean**	**STD**	**Max**	t_{mea}	**Min**	**Mean**	**STD**	**Max**	t_{mea}	**Min**	**Mean**	**STD**	**Max**	t_{mea}
Example 1	**4.7074**	**4.7074**	**0**	**4.7074**	**<1**	**4.7074**	**4.7074**	**0**	**4.7074**	**<1**	**4.7074**	**4.7074**	**0**	**4.7074**	**<1**
Example 2	3.1162	3.1694	0.0478	3.2467	352s	3.1162	3.1291	0.0522	3.2282	1130s	**3.2452**	**3.2463**	**0.0005**	**3.2467**	**186s**
Example 3	3.5855	3.7107	0.1243	3.8282	933s	3.5855	3.6649	0.1411	3.9738	1644s	**3.8282**	**4.0746**	0.1274	**4.4406**	**788 s**
Example 4	2.9096	3.0663	0.1252	3.2206	1172s	2.4375	2.9607	0.3082	3.2325	3960s	**3.2169**	**3.4577**	0.2154	**3.8991**	**2030s**

Table 3. Results for examples 1, 2, 3 and 4

In order to compare the performance of the three algorithms, the stopping criterion is the number of evaluated solutions. The computation time of the ACS algorithm, for the same number of evaluated solutions, is higher than that of the other algorithms. This is because the ACS algorithm constructs an *entire solution* (*i.e.*, selects versions and number of machines for each sub-system), at each iteration and for each ant. It implies that a complete loop is used. Thus, each solution construction requires considerable computation time. On the other hand, when the ACS is coupled with the improvement procedure, each generated solution by an ant can be improved by evaluating the neighbour solutions while carrying out *minor changes in the current solution*. Consequently, since it does *not require a complete construction of the solution*, the computation time is decreased. On the other hand, the simulated annealing algorithm constructs the solutions by making minor changes in the current solutions, requiring less computation time than the ACS algorithm when coupled or not with the improvement procedure.

Additional tests

A set of 10 test instances are also randomly generated for n = 20 and used to evaluate the performance of the proposed algorithms. Note that the parameters used for these 10 test instances are those set by using example 4 as a typical problem (see Table 3, for n = 20). By running the algorithms without further tuning on the 10 test instances, we avoid any parameters over-fitting.

The proposed algorithms are evaluated in terms of solutions quality. For each instance, five trials are performed. It has been observed again for these randomly generated instances that the ACS coupled with the Improvement procedure (ACS-I) out-performs ACS and SA algorithms.

5. Conclusion

Three optimal design problems were studied in this chapter. The first problem is related to the reliability optimization of series systems with multiple-choice and budget constraints. The second problem concerns the redundancy allocation problem of series-parallel systems, while the third deals with the selection of machines and buffers in unreliable series-parallel production lines. As the formulated problems are complicated combinatorial optimization ones, an exhaustive examination of all possible solutions is not realistic, considering reasonable time limitations. Because of this, we developed efficient heuristics to solve the formulated problems. This heuristic was inspired by the ant system meta-heuristic. The experimental results showed that the optimal or nearly optimal solutions are obtained very quickly. Through several numerical examples, the effectiveness of HACO with respect to the quality of solutions and the computing time will be discussed by performing comparisons with others approaches based on mate-heuristics.

6. References

Ait-Kadi D, Nourelfath M. Availability optimization of fault-tolerant systems. *Int Conf Ind Engng Prod Manage* (IEPM'2001) Quebec 2001; August. [1]

Dorigo M, Stutzle T. The ant colony optimization metaheuristic: Algorithms, applications and advances. *Handbook of Metaheuristics* 2001: F. Glover and G. Kochenberger. [2]

Garey MR, Johnson DS. Computers and Intractability. San Francisco:Freeman, 1979. [3]

Liang YC, Smith AE. An ant system approach to redundancy allocation. *Proceedings of the 1999 Congress on Evolutionary Computation*, 1999 (CEC 99);2: 1999 -1484. [4]

Nauss RM. The 0-1 knapsack problem with multiple choice constraints. *European Journal of Operational Research* 1978. p 121-131. [5]

Nourelfath M, Nahas N. Quantized Hopfield networks for reliability optimization. Reliab *Engng Sys Safety*, Volume 81, Issue 2, pages 191-196. [6]

Sinha P, Zoltners AA. The multiple choice knapsack problem. *Operations Research* 1979. p 503-515. [7]

Sung CS, Lee HK. A branch-and-bound approach for spare unit allocation in a series system. *European Journal of Operational Research* 1994. p 217-232. [8]

Sung CS, Cho YK. Reliability optimization of a series system with multiple-choice and budget constraints. *European Journal of Operational Research* 2000. p 159-171. [9]

Yang S, Dingwei W. Constraint satisfaction adaptive neural network and heuristics combined approaches for generalized job-shop scheduling. *IEEE Transactions on Neural Networks* 2000; ll(2):474-486. [10]

Nahas N, Nourelfath M. Ant system for reliability optimization of a series system with multiple-choice and budget constraints. *Reliab Eng Syst Saf* 2005;87(1):1-12. [11]

Bellman RE, Dreyfus E. Dynamic programming and reliability of multi-component devices. *Operations Research* 1958;6: 200-206. [12]

Besteo MD, Stutzle T, Dorigo M. Ant colony optimization for the total weighted tardiness problem. *Proc. 6th Int. Conf. Parallel Problem Solving From Nature* (PPSN VI), Berlin, 2000; pp. 611-620. [13]

Bulfin RL, Liu CY. Optimal allocation of redundant components for large systems. *IEEE Transactions on Reliability* 1985;34: 241-247. [14]

Bullnheimer B, Hartl RF, Strauss C. Applying the Ant System to the vehicle Routing problem. *2nd Metaheuristics International Conference* (MIC-97), Sophia-Antipolis, France, 1997; pp. 21-24. [15]

Burke EK, Bykov Y, Newall JP, Petrovic S. A New Local Search Approach with Execution Time as an Input Parameter. *Computer Science Technical Report No.* NOTTCS-TR-2002-3. School of Computer Science and Information Technology. University of Nottingham. [16]

Chern MS. On the computational complexity of reliability redundancy allocation in a series system. *Operations Research Letter* 1992; 11:309-15. [17]

Coit DW, Smith AE. Reliability Optimization of Series-Parallel Systems Using a Genetic Algorithm. *IEEE Transactions on Reliability* 1996;45(2): 254-260. [18]

Costa D, Hertz A. Ants can color graphs. *J. Oper. Res. Soc.* 1997; 48: 295-305. [19]

Dorigo M. Optimization, Learning and Natural Algorithms. *Ph.D Thesis*, Politecnico di Milano, Italy, 1992. [20]

Dorigo M, Maniezzo V, Colorni A. The Ant System: Optimization by a colony of cooperating agents. *IEEE Transactions on Systems*, Man and Cybernetics- Part B 1996; [21]

Dorigo M, Gambardella LM. Ant colonies for the traveling salesman problem. *BioSystems* 1997;43: 73-81. [22]

Dorigo M, Gambardella LM. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation* 1997; 1(1): 53-66. [23]

Di Caro G, Dorigo M. Mobile Agents for Adaptive Routing. *Proceedings for the 31st Hawaii International Conference on System Sciences*, Big Island of Hawaii, January 6-9, 1998, pp. 74-83. [24]

Fyffe DE, Hines WW, Lee NK. System Reliability Allocation And a Computational Algorithm. *IEEE Transactions on Reliability* 1968; vol. R-17(2):64-69. [25]

Gambardella LM, Taillard E, Dorigo M. Ant Colonies for the Quadratic Assignment Problem. *Journal of the Operational Research Society* 1999; 50:167-176. [26]

Gen M, Ida K, Tsujimura Y, Kim CE. Large scale 0-1 fuzzy goal programming and its application to reliability optimization problem. *Computers & Industrial Engineering* 1993; 24: 539-549. [27]

Ghare M, Taylor RE. Optimal redundancy for reliability in series system. *Operations research* 1969;17:838-847. [28]

Hsieh YC. A linear approximation for redundant reliability problems with multiple components choices. *Computers and Industrial Engineering* 2002; 44:91-103. [29]

Glover F. Future paths for integer programming and links to artificial intelligence. *Computers & Operations Research* 1986;13:533-549. [30]

Kulturel-Konak S, Smith AE, Coit D.W. Efficiently solving the redundancy allocation problem using tabu search. *HE transactions* 2003;35: 515-526. [31]

Kuo W, Prasad VR. An annotatedoverview of system-reliability optimization. *IEEE Transactions on Reliability* 2000;49(2): 176-87. [32]

Levitin G, Lisnianski A, Ben-Haim H, Elmakis D. Redundancy optimization for series-parallel multi-state systems. *IEEE Transactions on Reliability* 1998;47(2): 165-172. [33]

Misra KB, Sharma U. An Efficient Algorithm to Solve Integer-Programming Problems Arising in System-Reliability Design. *IEEE Transactions on Reliability* 1991;40(1): 81-91. [34]

Nakagawa Y, Miyazaki S. Surrogate Constraints Algorithm for Reliability Optimization Problems with Two Constraints. *IEEE Transactions on Reliability* 1981;R-30(2): 175-180. [35]

Nourelfath M, Nahas N, Ait-Kadi D. Optimal design of series production lines with unreliable machines and finite buffers. *Journal of Quality in Maintenance Engineering* 2005;11(2): 121-138. [36]

Painton L, Campbell J. Genetic Algorithms in Optimization of System Reliability. *IEEE Transactions on Reliability* 1995;44(2): 172-178. [37]

Schoofs L, Naudts B. Ant colonies are good at solving constraint satisfaction problems. *Proc. 2000 Congress on Evolutionary Computation*, San Diego, CA, July 2000, pp. 1190-1195. [38]

Tillman FA, Hwang CL, Kuo W. Optimization techniques for system reliability with redundancy- a review. *IEEE Transaction on Reliability* 1977;R-26(3): 147-155. [39]

Tillman FA, Hwang C-L, Kuo W. Determining Component Reliability and Redundancy for Optimum System Reliability. *IEEE Transactions on Reliability* 1977;R-26(3): 162-165. [40]

Yalaoui A, Chatelet E, Chu C. A New Dynamic Programming Method for Reliability and Redundancy Allocation in a Parallel-Series System. *IEEE transactions on Reliability* 2005; 54 (2):254-261. [41]

Yokota T, Gen M, Ida K. System reliability of optimization problems with several failure modes by genetic algorithm. *Japanese Journal of Fuzzy Theory and Systems* 1995; 7(l):117-35. [42]

Yokota T, Gen M, Li YX. Genetic algorithm for nonlinear mixed-integer programming and its applications. *Computers and Industrial Engineering* 1996;30(4):905-17. [43]

Wagner IA, Bruckstein AM. Hamiltonian(t)—an ant inspired heuristic for recognizing Hamiltonian graphs. *Proc. 1999 Congress on Evolutionary Computation*, Washington, D.C., July 1999, pp. 1465-1469. [44]

Dallery, Y. and Gershwin, S.B., 1992, Manufacturing Flow Line Systems: A Review of Models and Analytical Results, Queueing Systems theory and Applications, *Special Issue on Queueing Models of Manufacturing Systems*, 12(1-2), 3-94. [45]

Gershwin, S.B. and Schor, J.E., 2000, Efficient Algorithms for Buffer Space Allocation, *Annals of Operations Research*, 93, 117-144. [46]

Buzacott, J.A. 1967, Automatic Transfer Lines with Buffer Stocks, *International Journal of Production Research*, 5(3), 182-200. [47]

Buzacott, J.A., 1968, Prediction of Efficiency of Production Systems without Internal Storage, *International Journal of production Research*, 6(3), 173-188. [48]

Gershwin, S.B., 1987, An Efficient Decomposition Algorithm for The Approximate Evaluation of Tandem Queues with Finite Storage Space and Blocking, *Operations Research*, 35, 291-305. [49]

Kirckpatrick, S., GERLATT C.D. Jr and VECCHI M.P., 1983, Optimization by Simulated Annealing, *Science*, 220, 671-680. [50]

DALLERY, Y., DAVID, R. and Xffi, X.L., 1988, An Efficient Algorithm for Analysis of Transfer Lines with Unreliable machines and Finite Buffers, *HE transactions*, 20(3), 280-283. [51]

Burman, M.H., 1995, New Results in Flow Line Analysis, *Ph. D. Thesis*, MIT, Cambridge MA. [52]

Le Bihan, H. and Dallery, Y., 1997, Homogenisation Techniques for the Analysis of Production Lines with Unreliable Machines Having Different Speeds, *European Journal of Control*, 3, 200-215. [53]

Liu X-G and Buzacott, J.A., 1990, Approximate models of assembly systems with finite banks, *European Journal of Operational Research*, 45, 145-154. [54]

N. Nahas, D., M. Nourelfath et Ait-Kadi (2007). Coupling ant colony and the degraded ceiling algorithm for the redundancy allocation problem of series-parallel systems. *Reliability Engineering and System Safety*. Volume 92, Issue 2, Pages 211-222. [55]

N. Nahas, M. Nourelfath et D. Ait-Kadi (2006). Selecting machines and buffers in unreliable series-parallel production lines. *International Journal of Production Research*. (Submitted). [56]

PENCUS, M., 1970, A monte carlo method for the approximate solution of certain types of constrained optimization problems, *Oper. Rese*., 18, 1225-1228. [57]

SPINELLIS, D. and PAPADOPOULOS, C.T., 2000, A simulated annealing approach for buffer allocation in reliable production lines. *Annals of Operations Research*, 93, 373-384. [58]

7. Appendix

		Version 1	Version 2	Version 3	Version 4
Machine 1	μ, λ, P	0.0735, 0.0121, 2.2238	0.1470, 0.0289, 2.3338	0.0438, 0.0075, 2.3764	0.0392, 0.0068, 2.4643
	Cost($)	5	10	13	15
Machine 2	μ, λ, P	0.1557, 0.0530, 2.027	0.1588, 0.0446, 2.0396	0.1495, 0.0375, 2.1153	0.0358, 0.0082, 2.1273
	Cost($)	10	15	20	25
Machine 3	μ, λ, P	0.0521, 0.0053, 2.1849	0.0447, 0.0051, 2.2222	0.1383, 0.0205, 2.2924	0.0523, 0.0062, 2.3265
	Cost($)	10	12	15	20
Machine 4	μ, λ, P	0.0336, 0.0023, 2.1568	0.0916, 0.0037, 2.3087	0.1007, 0.0044, 2.3541	0.1074, 0.0055, 2.4084
	Cost($)	9	15	23	30

Table A. 1. Data for example 1

		Version 1	Version 2	Version 3
Buffer 1	Capacity, Cost($)	30, 5	40, 8	55, 15
Buffer 2	Capacity, Cost($)	45,10	60, 20	65, 25
Buffer 3	Capacity, Cost($)	35, 5	50, 10	60, 18

Table A.2. Data for example 1

		Version 1	Version 2	Version 3	Version 4	Version 5
Machine 1	μ, λ, P	0.2645, 0.0438, 3.1145	0.1544, 0.0268, 3.3136	0.2468, 0.0433, 3.3426	0.1593, 0.0269, 3.3977	0.3688, 0.0622, 3.5288
	Cost($)	5	7	10	12	13
Machine 2	μ, λ, P	0.2085, 0.0303, 2.9765	0.2171, 0.0283, 3.1229	0.3707, 0.0522, 3.4287	0.195, 0.0315, 3.5041	0.2525, 0.0339, 3.5389
	Cost($)	7	10	12	13	14
Machine 3	μ, λ, P	0.2351, 0.1242, 2.5327	0.2691, 0.0861, 2.783	0.1953, 0.0574, 2.9909	0.2192, 0.0484, 3.07	0.199, 0.0547, 3.2178
	Cost($)	10	11	12	15	17
Machine 4	μ, λ, P	0.1528, 0.0289, 2.9659	0.2796, 0.0535, 3.0089	0.2921, 0.0617, 3.2554	0.3878, 0.0798, 3.3826	0.1809, 0.0318, 3.398
	Cost($)	5	8	9	10	14
Machine 5	μ, λ, P	0.2314, 0.0242, 3.1254	0.2567, 0.0254, 3.1522	0.2103, 0.0175, 3.4252	0.1982, 0.021, 3.4279	0.1957, 0.0155, 3.5828
	Cost($)	20	22	23	25	26
Machine 6	μ, λ, P	0.1599, 0.0108, 3.2422	0.3256, 0.0221, 3.292	0.234, 0.018, 3.481	0.218, 0.0148, 3.5796	0.2695, 0.0196, 3.629
	Cost($)	20	22	23	25	26
Machine 7	μ, λ, P	0.2612, 0.0323, 3.0859	0.1718, 0.0217, 3.3243	0.1827, 0.0221, 3.3351	0.1552, 0.0172, 3.6137	0.3862, 0.047, 3.6151
	Cost($)	10	13	15	16	17
Machine 8	μ, λ, P	0.2287, 0.0108, 3.6734	0.3794, 0.015, 3.7013	0.1845, 0.0072, 3.818	0.2565, 0.0088, 3.8469	0.2403, 0.0096, 3.9015
	Cost($)	10	11	12	14	15
Machine 9	μ, λ, P	0.2494, 0.0153, 3.4925	0.2221, 0.0115, 3.5739	0.3126, 0.0198, 3.6441	0.1941, 0.0113, 3.7465	0.2547, 0.0126, 3.8411
	Cost($)	5	6	8	10	11
Machine 10	μ, λ, P	0.2285, 0.0051, 3.4271	0.3272, 0.0074, 3.6841	0.3721, 0.0124, 3.7248	0.1863, 0.0051, 3.828	0.3115, 0.0097, 3.9345
	Cost($)	15	16	18	20	21

Table A.3. Data for example 2

		Version 1	Version 2	Version 3	Version 4
Buffer 1	Capacity, Cost($)	40, 5	55, 8	70, 14	80, 20
Buffer 2	Capacity, Cost($)	30, 5	40, 8	50, 14	65, 20
Buffer 3	Capacity, Cost($)	30, 7	40, 10	45, 15	60, 18
Buffer 4	Capacity, Cost($)	45, 10	55, 15	60, 19	70, 23
Buffer 5	Capacity, Cost($)	35, 12	50, 15	67, 20	70, 30
Buffer 6	Capacity, Cost($)	40, 10	50, 15	65, 19	70, 23
Buffer 7	Capacity, Cost($)	50, 5	65, 8	75, 14	85, 20
Buffer 8	Capacity, Cost($)	30, 15	55, 20	65, 24	80, 28
Buffer 9	Capacity, Cost($)	30, 10	35, 15	40, 20	45, 25

Table A.4. Data for example 2

		Version 1	Version 2	Version 3	Version 4	Version 5
Machine 1	μ, λ, P	0.2304, 0.1166, 2.6313	0.1973, 0.0972, 2.676	0.2139, 0.1336, 2.7256	0.0714, 0.0447, 2.8628	0.106, 0.0448, 2.9109
	Cost($)	5	7	10	12	13
Machine 2	μ, λ, P	0.1898, 0.0507, 3.1064	0.0923, 0.0205, 3.5717	0.1508, 0.0344, 3.7054	0.0792, 0.0204, 3.7109	0.0912, 0.019, 3.7525
	Cost($)	7	10	12	13	14
Machine 3	μ, λ, P	0.17, 0.025, 3.4031	0.0788, 0.0118, 3.4336	0.2261, 0.0327, 3.5206	0.0609, 0.0087, 3.8141	0.0615, 0.0092, 4.0129
	Cost($)	10	11	12	15	17
Machine 4	μ, λ, P	0.1093, 0.0404, 2.81	0.0682, 0.0222, 2.9181	0.1566, 0.0444, 3.0724	0.1804, 0.0562, 3.4688	0.1919, 0.0516, 3.667
	Cost($)	5	8	9	10	14
Machine 5	μ, λ, P	0.105, 0.0202, 3.4092	0.2216, 0.0352, 3.4678	0.1866, 0.0309, 3.6719	0.0751, 0.0113, 3.9665	0.0801, 0.0126, 3.9949
	Cost($)	20	22	23	25	26
Machine 6	μ, λ, P	0.1022, 0.0103, 3.5011	0.0719, 0.0073, 3.6115	0.1238, 0.0127, 3.652	0.0806, 0.0085, 3.6843	0.1755, 0.019, 4.243
	Cost($)	20	22	23	25	26
Machine 7	μ, λ, P	0.2373, 0.0236, 3.6365	0.1379, 0.0134, 4.1087	0.1587, 0.0156, 4.2079	0.1497, 0.0143, 4.212	0.1063, 0.0103, 4.2268
	Cost($)	10	13	15	16	17
Machine 8	μ, λ, P	0.0804, 0.0091, 3.4941	0.1803, 0.0206, 3.5102	0.1389, 0.0188, 3.6295	0.0767, 0.0089, 3.7977	0.1837, 0.0253, 4.0229
	Cost($)	10	11	12	14	15
Machine 9	μ, λ, P	0.1958, 0.0157, 3.6803	0.158, 0.0112, 3.8792	0.1185, 0.0094, 4.006	0.0979, 0.0077, 4.1912	0.0717, 0.0054, 4.2743
	Cost($)	5	6	8	10	11
Machine 10	μ, λ, P	0.166, 0.0147, 4.0969	0.0918, 0.0083, 4.1199	0.0975, 0.0082, 4.1955	0.0949, 0.0086, 4.2948	0.1807, 0.0148, 4.313
	Cost($)	15	16	18	20	21
Machine 11	μ, λ, P	0.1157, 0.008, 3.7438	0.1402, 0.0095, 3.8245	0.087, 0.0059, 3.9554	0.0641, 0.0045, 3.9678	0.0964, 0.0065, 4.0083
	Cost($)	8	13	15	16	17
Machine 12	μ, λ, P	0.079, 0.0013, 3.7965	0.0952, 0.0023, 4.019	0.2368, 0.0047, 4.2889	0.1972, 0.0044, 4.5438	0.0787, 0.0015, 4.5666
	Cost($)	9	11	12	15	17
Machine 13	μ, λ, P	0.1524, 0.1524, 1.9293	0.1309, 0.1309, 2.0227	0.1779, 0.1779, 2.1252	0.0762, 0.0762, 2.149	0.0983, 0.0983, 2.1636
	Cost($)	5	7	10	12	13
Machine 14	μ, λ, P	0.1997, 0.0077, 3.8336	0.0854, 0.0035, 4.1152	0.0791, 0.0034, 4.1519	0.0972, 0.0032, 4.1718	0.0688, 0.0019, 4.2924
	Cost($)	10	12	13	15	16
Machine 15	μ, λ, P	0.1573, 0.0099, 3.6318	0.0904, 0.0053, 3.7954	0.1232, 0.008, 4.0276	0.0637, 0.0039, 4.2929	0.1181, 0.0066, 4.3657
	Cost($)	5	8	10	12	15

Table A.5. Data for example 3

		Version 1	Version 2	Version 3	Version 4
Buffer 1	Capacity, Cost($)	40, 5	55, 8	70, 14	80, 20
Buffer 2	Capacity, Cost($)	30, 5	40, 8	50, 14	65, 20
Buffer 3	Capacity, Cost($)	30, 7	40, 10	45, 15	60, 18
Buffer 4	Capacity, Cost($)	45, 10	55, 15	60, 19	70, 23
Buffer 5	Capacity, Cost($)	35, 12	50, 15	67, 20	70, 30
Buffer 6	Capacity, Cost($)	40, 10	50, 15	65, 19	70, 23
Buffer 7	Capacity, Cost($)	50, 5	65, 8	75, 14	85, 20
Buffer 8	Capacity, Cost($)	30, 15	55, 20	65, 24	80, 28
Buffer 9	Capacity, Cost($)	30, 10	35, 15	40, 20	45, 25
Buffer 10	Capacity, Cost($)	30, 7	40, 10	45, 15	60, 18
Buffer 11	Capacity, Cost($)	35, 10	55, 15	60, 19	70, 23
Buffer 12	Capacity, Cost($)	30, 12	40, 15	67, 20	70, 30
Buffer 13	Capacity, Cost($)	40, 10	50, 15	65, 19	70, 23
Buffer 14	Capacity, Cost($)	40, 5	55, 8	65, 14	85, 20

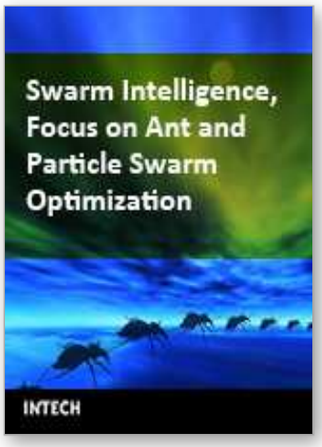
Table A.6. Data for example 3

		Version 1	Version 2	Version 3	Version 4	Version 5
Machine 1	μ, λ, P	0.0185, 0.0113, 1.7486	0.0813, 0.0115, 2.152	0.0931, 0.018, 2.2771	0.0353, 0.0072, 2.3781	0.0245, 0.0024, 2.6762
	Cost($)	5	7	10	12	13
Machine 2	μ, λ, P	0.0371, 0.0178, 1.9237	0.0202, 0.0042, 2.0775	0.0295, 0.0040, 2.3179	0.0303, 0.0019, 2.3961	0.0189, 0.0006, 2.7135
	Cost($)	7	10	12	13	14
Machine 3	μ, λ, P	0.0382, 0.0114, 1.8927	0.1223, 0.0204, 2.2257	0.128, 0.0098, 2.2937	0.0297, 0.0042, 2.2937	0.0186, 0.001, 2.8012
	Cost($)	10	11	12	15	17
Machine 4	μ, λ, P	0.0823, 0.0254, 1.9109	0.0585, 0.0226, 2.1123	0.0231, 0.001, 2.4097	0.0219, 0.0032, 2.4536	0.1037, 0.0205, 2.4773
	Cost($)	5	8	9	10	14
Machine 5	μ, λ, P	0.068, 0.0257, 1.8233	0.076, 0.0117, 2.1325	0.0611, 0.0022, 2.3506	0.08, 0.0134, 2.4409	0.1092, 0.0026, 2.7139
	Cost($)	20	22	23	25	26
Machine 6	μ, λ, P	0.0295, 0.0044, 2.2169	0.035, 0.0046, 2.5357	0.04804, 0.0068, 2.5703	0.033, 0.001, 2.7065	0.0354, 0.0012, 2.8585
	Cost($)	20	22	23	25	26
Machine 7	μ, λ, P	0.0353, 0.0139, 1.8043	0.03, 0.0029, 2.2432	0.1072, 0.0044, 2.3552	0.0294, 0.0006, 2.3913	0.0285, 0.0026, 2.5494
	Cost($)	10	13	15	16	17
Machine 8	μ, λ, P	0.0994, 0.0312, 1.8479	0.0359, 0.0072, 2.2335	0.0518, 0.0052, 2.2525	0.0285, 0.0007, 2.5563	0.0226, 0.0012, 2.7477
	Cost($)	10	11	12	14	15
Machine 9	μ, λ, P	0.0207, 0.0045, 2.1397	0.0988, 0.0286, 2.2173	0.0179, 0.0029, 2.3142	0.0891, 0.0126, 2.5253	0.0212, 0.0008, 2.5373
	Cost($)	5	6	8	10	11
Machine 10	μ, λ, P	0.094, 0.0142, 2.1576	0.1087, 0.013, 2.2653	0.0588, 0.0098, 2.4437	0.0931, 0.0058, 2.5515	0.0225, 0.0014, 2.6952
	Cost($)	15	16	18	20	21
Machine 11	μ, λ, P	0.0377, 0.0053, 2.152	0.1138, 0.0111, 2.3944	0.1134, 0.0107, 2.437	0.0216, 0.0012, 2.6218	0.1068, 0.0135, 2.6254
	Cost($)	8	13	15	16	17
Machine 12	μ, λ, P	0.0181, 0.0078, 1.8908	0.1131, 0.0068, 2.3775	0.0182, 0.0025, 2.4526	0.0572, 0.0042, 2.4693	0.1201, 0.0086, 2.4944
	Cost($)	9	11	12	15	17
Machine 13	μ, λ, P	0.0612, 0.006, 2.3013	0.0612, 0.0027, 2.5733	0.0594, 0.0055, 2.6149	0.0321, 0.0021, 2.6493	0.1287, 0.0065, 2.6888
	Cost($)	5	7	10	12	13
Machine 14	μ, λ, P	0.0281, 0.0123, 1.8713	0.0317, 0.0029, 2.3124	0.0192, 0.0036, 2.3596	0.0613, 0.0073, 2.6254	0.0764, 0.0072, 2.688
	Cost($)	10	12	13	15	16
Machine 15	μ, λ, P	0.0279, 0.0071, 2.06	0.0296, 0.0054, 2.1132	0.1134, 0.0182, 2.3418	0.1072, 0.0046, 2.5759	0.0311, 0.0012, 2.7956
	Cost($)	5	8	10	12	15
Machine 16	μ, λ, P	0.0307, 0.0034, 2.3659	0.0339, 0.0035, 2.5748	0.1291, 0.011, 2.6265	0.017, 0.0012, 2.6318	0.1279, 0.0044, 2.7025
	Cost($)	5	8	9	10	14
Machine 17	μ, λ, P	0.0888, 0.0449, 1.7617	0.0831, 0.0135, 2.3343	0.0346, 0.0029, 2.3881	0.0248, 0.0017, 2.3942	0.0175, 0.0005, 2.8557
	Cost($)	20	22	23	25	26
Machine 18	μ, λ, P	0.0316, 0.0097, 1.9782	0.03, 0.0022, 2.3827	0.0659, 0.0053, 2.5769	0.0704, 0.0078, 2.6572	0.0215, 0.0013, 2.775
	Cost($)	20	22	23	25	26
Machine 19	μ, λ, P	0.0615, 0.0183, 2.2777	0.0334, 0.0076, 2.3789	0.0172, 0.0026, 2.3951	0.123, 0.0084, 2.6452	0.032, 0.0009, 2.6673
	Cost($)	10	13	15	16	17
Machine 20	μ, λ, P	0.0244, 0.0073, 1.9532	0.0715, 0.0075, 2.4901	0.1182, 0.0061, 2.6246	0.0291, 0.0019, 2.644	0.0719, 0.0057, 2.6495
	Cost($)	10	11	12	14	15

Table A.7. Data for example 4

		Version 1	Version 2	Version 3	Version 4
Buffer 1	Capacity, Cost($)	40, 5	55, 8	70, 14	80, 20
Buffer 2	Capacity, Cost($)	30, 5	40, 8	50, 10	65, 13
Buffer 3	Capacity, Cost($)	30, 7	40, 10	45, 11	60, 15
Buffer 4	Capacity, Cost($)	45, 10	55, 12	60, 15	70, 20
Buffer 5	Capacity, Cost($)	35, 12	50, 15	67, 20	70,30
Buffer 6	Capacity, Cost($)	40, 10	50, 11	65, 13	70, 14
Buffer 7	Capacity, Cost($)	50, 5	65, 8	75, 10	85, 13
Buffer 8	Capacity, Cost($)	30, 17	55, 20	65, 24	80, 28
Buffer 9	Capacity, Cost($)	30, 10	35, 12	40, 15	45, 16
Buffer 10	Capacity, Cost($)	30, 7	40, 10	45, 15	60, 18
Buffer 11	Capacity, Cost($)	35, 10	55, 12	60, 15	70, 18
Buffer 12	Capacity, Cost($)	30, 12	40, 15	67, 20	70, 23
Buffer 13	Capacity, Cost($)	40, 10	50, 15	65, 19	70, 23
Buffer 14	Capacity, Cost($)	40, 5	55, 8	65, 13	85, 17
Buffer 15	Capacity, Cost($)	50, 5	65, 8	75, 12	85, 15
Buffer 16	Capacity, Cost($)	30, 15	55, 16	65, 18	80, 21
Buffer 17	Capacity, Cost($)	30, 10	35, 15	40, 20	45, 25
Buffer 18	Capacity, Cost($)	30, 7	40, 10	45, 12	60, 15
Buffer 19	Capacity, Cost($)	35, 10	55, 15	60, 19	70, 23

Table A. 8. Data for example 4



Swarm Intelligence, Focus on Ant and Particle Swarm Optimization
Edited by FelixT.S.Chan and Manoj KumarTiwari

ISBN 978-3-902613-09-7
Hard cover, 532 pages
Publisher I-Tech Education and Publishing
Published online 01, December, 2007
Published in print edition December, 2007

In the era globalisation the emerging technologies are governing engineering industries to a multifaceted state. The escalating complexity has demanded researchers to find the possible ways of easing the solution of the problems. This has motivated the researchers to grasp ideas from the nature and implant it in the engineering sciences. This way of thinking led to emergence of many biologically inspired algorithms that have proven to be efficient in handling the computationally complex problems with competence such as Genetic Algorithm (GA), Ant Colony Optimization (ACO), Particle Swarm Optimization (PSO), etc. Motivated by the capability of the biologically inspired algorithms the present book on "Swarm Intelligence: Focus on Ant and Particle Swarm Optimization" aims to present recent developments and applications concerning optimization with swarm intelligence techniques. The papers selected for this book comprise a cross-section of topics that reflect a variety of perspectives and disciplinary backgrounds. In addition to the introduction of new concepts of swarm intelligence, this book also presented some selected representative case studies covering power plant maintenance scheduling; geotechnical engineering; design and machining tolerances; layout problems; manufacturing process plan; job-shop scheduling; structural design; environmental dispatching problems; wireless communication; water distribution systems; multi-plant supply chain; fault diagnosis of airplane engines; and process scheduling. I believe these 27 chapters presented in this book adequately reflect these topics.

How to reference
In order to correctly reference this scholarly work, feel free to copy and paste the following:

Nabil Nahas, Mustapha Nourelfath and Daoud Ait-Kadi (2007). Ant Colonies for Performance Optimization of Multi-components Systems Subject to Random Failures, Swarm Intelligence, Focus on Ant and Particle Swarm Optimization, FelixT.S.Chan and Manoj KumarTiwari (Ed.), ISBN: 978-3-902613-09-7, InTech, Available from: http://www.intechopen.com/books/swarm_intelligence_focus_on_ant_and_particle_swarm_optimization/ant_colonies_for_performance_optimization_of_multi-components_systems_subject_to_random_failures

INTECH
open science | open minds

InTech Europe
University Campus STeP Ri
Slavka Krautzeka 83/A
51000 Rijeka, Croatia
Phone: +385 (51) 770 447
Fax: +385 (51) 686 166
www.intechopen.com

InTech China
Unit 405, Office Block, Hotel Equatorial Shanghai
No.65, Yan An Road (West), Shanghai, 200040, China
中国上海市延安西路65号上海国际贵都大饭店办公楼405单元
Phone: +86-21-62489820
Fax: +86-21-62489821

IntechOpen