Journal of Heuristics, 10: 269–292, 2004

A Distributed Evolutionary Simulated Annealing Algorithm for Combinatorial Optimisation Problems

M. EMIN AYDIN*
University of the West of England, Faculty of Computing, Engineering and Mathematical Sciences, Frenchey Campus, Colharbour Lane, Bristol, BS16 1QY, UK
email: me-aydin@uwe.ac.uk

TERENCE C. FOGARTY
London South Bank University, Faculty of Business, Computing and Information Management, 103 Borough Road, London, SE1 0AA, UK
email: fogarttc@lsbu.ac.uk

Submitted in October 2003 and accepted by Enrique Alba in March 2004 after 1 revision

Abstract

In this paper, the Evolutionary Simulated Annealing (ESA) algorithm, its distributed implementation (dESA) and its application to two combinatorial problems are presented. ESA consists of a population, a simulated annealing operator, instead of the more usual reproduction operators used in evolutionary algorithms, and a selection operator. The implementation is based on a multi island (agent) system running on the Distributed Resource Machine (DRM), which is a novel, scalable, distributed virtual machine based on Java technology. As WAN/LAN systems are the most common multi-machine systems, dESA implementation is based on them rather than any other parallel machine. The problems tackled are well-known combinatorial optimisation problems, namely, the classical job-shop scheduling problem and the uncapacitated facility location problem. They are difficult benchmarks, widely used to measure the efficiency of metaheuristics with respect to both the quality of the solutions and the central processing unit (CPU) time spent. Both applications show that dESA solves problems finding either the optimum or a very near optimum solution within a reasonable time outperforming the recent reported approaches for each one allowing the faster solution of existing problems and the solution of larger problems.

Key Words: distributed evolutionary simulated annealing, multi-agent systems, job-shop scheduling, uncapacitated facility location, distributed resource machine

1. Introduction

Simulated annealing (SA) is a stochastic heuristic algorithm, which searches the solution space using a stochastic hill climbing process. Because of its ease of use, SA is an extremely popular method for solving large and practical problems such as job-shop scheduling, timetabling and travelling salesman etc. However, SA has two drawbacks:

*Author to whom all correspondence should be addressed.

being trapped by local minima or taking too long to find a reasonable solution. In order to overcome these drawbacks, researchers have either hybridised SA with other heuristics such as the genetic algorithms or implemented SA as parallel algorithms. The main aim is to avoid local minima traps and/or to have faster convergence. On the one hand, many hybrid approaches combining the genetic algorithm (GA) with SA have been implemented to take advantage of a diverse population provided by the GA and hill climbing provided by SA (Jeong and Lee, 1996; Kolonko, 1999; Hung et al., 2001; Wang and Zheng, 2001; Wong, 2001). On the other hand, a number of parallel SA (PSA) algorithms have been developed to overcome its speed problem—since the late 1980s attempts have been made to parallelise the SA's inner process through parallel computing. Greening (1990), Kim, Jang, and Kim (1991), Ferreira and Zerovnik (1993), Boissin and Lutton (1993) and Voogd, Sloot, and Dantzig (1994) have all discussed PSA algorithms for various combinatorial optimisation problems. Logically, the SA algorithms can mainly be parallelised in two ways. One allows the partitioning of the inner loop of SA in which several permutations are done just before cooling the temperature (Bongiovanni, Crescenzi, and Guerra, 1995; Vales-Alsano et al., 2003). The other is based on parallelising the data. The infrastructure of parallelisation is mainly the technology of parallel machines: SIMD, MIMD etc. Those resources are utilised to process the tasks in parallel either by algorithmic parallelisation decomposing the inner loops or by Multiple Independent Run (MIR) methods (Bhandarkar et al., 1998; Chu, Deng, and Reinitz, 1999; Bevilacqua, 2002) in which independent runs are launched at the same time with different initial solutions on multiple resources. MIR provided by multi threading and the communications needed between threads (independent runs) are handled by using Massage Passing Interfaces (MPI), which are developed within various programming environments. Since the hardware of parallel machines (multi-processor machines) is unified and homogenised, it is easy to parallelise and build fast communication between the independent runs making the algorithmic parallelisation of SA affordable. However, it is not easy to build in fast communication for an infrastructure based on the multiple heterogeneous machines that exist in local/wide area networks.

This work on ESA is relevant to PSA algorithms, as the aim is to run a distributed SA algorithm. Likewise, it is relevant to evolutionary algorithms (EAs), as it uses a SA operator and selection to operate on a population of solutions. Furthermore, EAs are inherently parallel and distributable since the genetic operators are working independently on different individuals of the population. In fact the algorithm presented in this paper is an EA, since a SA operator has been adopted instead of the standard crossover and mutation operators and a selection strategy. As mentioned before, it is easy to parallelise and distribute EAs as they are inherently parallel. There are three different models of distributed EAs: client-server model, island model, and diffusion model. For further information see Schmeck, Branke, and Kohlmorgen (2001) and Alba and Tomassini (2002). Since the client-server model is much centralised and the diffusion model requires very special hardware resource, we chose to implement our algorithm based on the island model.

The island model is easily implemented on and distributed over commonly used computer networks. The main idea of our work is to parallelise and distribute the ESA algorithm to improve its performance for combinatorial optimisation problems by partitioning a bigger

(main) population into small subparts and place each on a particular unit of resource (agent, in this case). The parallelisation and the distribution are carried out through multi agent (island) systems. We create a multi island system that works in parallel on a distributed computer system such as WAN or LAN and communicates via Internet connection. Each separate island is furnished with an ESA algorithm to evolve the population dispatched. By implementing in this way, each individual gets more opportunities to be selected across the whole search. The most relevant work to our approach is done by Yong, Lishan, and Evans (1995) suggesting an annealing evolution algorithm for function optimisation. A SA process is embedded into a generational evolutionary algorithm where the SA takes a very long time to finalise a single generation. On the other hand, our algorithm evaluates the individuals just after each separated SA operation so that one more genetic operator can be easily added to the algorithm. This provides our algorithm with more modularity and scalability over others but since they only applied their algorithms to function optimisation, it is difficult to do a comparison.

As mentioned before, we have tackled two very well known NP-hard combinatorial optimisation problems: the classical job shop scheduling (JSS) problem and the uncapacitated facility location (UFL) problem. The benchmarks undertaken are from the OR Library (Beasley, 1996), a collection of benchmarks for operations research (OR) studies. The distributed implementation of the ESA algorithm has been done as a multi island model to run on the Distributed Resource Machine (DRM), which is a novel scalable distributed problem-solving environment (Paechter et al., 2000). Each island has a separate ESA algorithm running in parallel with the others.

The organisation of the rest of the paper is as follows. We introduce the foundations of the ESA algorithm in Section 2. Then, we describe why and how to distribute ESA in Section 3. Afterwards, we present the job shop scheduling application of dESA in Section 4 and uncapacitated facility location application in Section 5. We discuss empirical results within each relevant section. The overall conclusion of the paper is in Section 6.

2. Evolutionary simulated annealing (ESA)

Simulated annealing can be viewed as a probabilistic decision making process in which a control parameter called temperature is employed to evaluate the probability of accepting an uphill move (in the case of minimisation). Suppose that s_n, s'_n and s_{n+1} denote the solution held (moved from) in the nth iteration, the solution moved to in the nth iteration, and the qualified solution for the $n + 1$th iteration, respectively. s'_n is yielded by moving from s_n by the way of the neighbourhood function. The new qualified solution is determined as follows:

$$s_{n+1} \leftarrow \begin{cases} s'_n & \Delta s < 0 \\ s'_n & e^{\frac{-\Delta s}{t_n}} > \rho \\ s_n & \text{otherwise} \end{cases}$$

where $\Delta s = s'_n - s_n$, ρ is the random number generated for making a stochastic decision for the new solution and t_n is the level of temperature (at the nth iteration), which is cooled through this optimisation process by a particular cooling function, $f(t_n)$. This means that, in order to make the new solution (s'_n) qualified for the next iteration, it must either be better than the old one (s_n) or at least the stochastic rule be satisfied. The stochastic rule, in which the probabilistic decision is made to prevent the optimisation process from sticking in possible local minima, is the main idea behind simulated annealing. The probability of making such a decision under the circumstances of a Δs at temperature, t_n, is denoted by $e^{-\Delta s/t_n}$. Clearly, as the temperature is decreased by $f(t_n)$, the probability of accepting a large decrease in solution quality decays exponentially toward zero according to the Boltzman distribution. Therefore, the final solution is near optimal when the temperature approaches zero.

As pointed out in the literature (Wong, 2001; Wang and Zhang, 2001), SA is able to converge to the optimum value, but it could be very expensive to get a desired solution in terms of computational time. For that reason, SA has been improved by hybridising it with other methods such as the genetic algorithms (Kolonko, 1999; Huang et al., 2001) or by parallelising the algorithm (Chu, Deng, and Reinitz, 1999; Bevilacqua, 2002). The evolutionary simulated annealing algorithm was developed to contribute to the progress in this direction. It offers an evolutionary process in which a shorter SA algorithm is substituted for the genetic operators of crossover and mutation to evolve a population of solutions. The SA algorithm is so compact that one can easily use it in any evolutionary process, where SA can manipulate the solutions selected from the population. This makes ESA easily implementable in various environments and together with different methods.

ESA adopts a SA as an evolutionary operator together with other evolutionary operators. We embedded a SA into an evolutionary algorithm, which does not contain any reproduction operators (crossover, mutation). The algorithm is sketched in figure 1. After initialisation and parameter setting, the algorithm repeats the following steps: (i) selects one individual subject to the running selection rule, (ii) operates on it with the SA operator, and (iii) evaluates whether to put it back into the population or not by a particular replacement rule. The details of SA operator are given in figure 2. All the usual elements of a typical simulated annealing algorithm are contained except that there are no inner repetitions, which are implemented during the acceptance stage before decreasing the level of temperature. In this case, the neighbourhood function works once to cool the temperature per iteration. For instance, the total number of moves per SA operation becomes 200 when the highest

Begin
⇨ *Initialise* the **population (*P*)**,
⇨ *Set* the **number of evaluations (*N*)**
Repeat:
- *Select* an **individual (p_n)**
- *Operate* by the **SA** and *get* the **new individual (p_n')**
- *Evaluate* the **new individual** for **replacement**

Until ***n*>=*N***
End.

Figure 1. A pseudo code for ESA algorithm.

Begin:

- *pick* one **feasible solution (p_n)**,
- *se*t the **highest temperature *(t)*,**
- *set* ***i=0*** and $p_i = p_n$

repeat:-

- *conduct* a move by **neighbourhood function** to get a **new solution (p_i')**
- *if* (p_i'-p_i)<0 then $p_{i+1} = p_i$'
- *else*
  - *generate* a **random number *(r)***
  - *if* ***exp(-(p_i'-p_i)/t)>r*** then $p_{i+1} = p_i$'
  - *else* $p_{i+1} = p_i$
  - *endif*
- *endif*
- ***t=f(t)*** *and* ***i=i+1***

until ***t*** reached a certain level (pre-defined)

- $p_n' = p_i$

End.

Figure 2. A pseudo code for SA operator.

temperature of 100 (t) is decayed by 0.955 ($f(t)$) per iteration until it reaches 0.01, as implemented for job shop scheduling problems (Aydin and Fogarty, 2002).

Obviously, the only operator running together with selection is the SA. Since the SA operator re-operates on particular solutions several times, the whole method works as if it is reheated every certain number of iterations. If we suppose that there is a single solution operated by this SA, it will become a multi-start (not multi-run[1]) algorithm that reruns repeatedly. Thus, the novelty of ESA can be viewed from two points of view: one is its multi-start property, and the other is its evolutionary approach. The multi-start property provides ESA with a more uniform distribution of random moves along the whole procedure and that helps to diversify the solutions. In fact, typical SA works in such a way that the search space is explored by exponentially distributed random moves, where each random move starts a new hill climbing process to reach the global minimum. However, the random moves are condensed in the earlier stages of the cooling process and the probability of having random moves goes down exponentially. Since it almost behaves like a hill climber in the later stages of the process, it becomes harder to escape from local minima then, especially, when it is applied to very difficult combinatorial optimisation problems, which have harder local minima. The idea is to distribute the random moves more uniformly than exponentially across the whole process.

Suppose that we have a problem that has a landscape, ℓ, and E_0 as one of very strict local minima. Furthermore, suppose we run a SA algorithm that sticks in E_0 under some initial conditions. Most of the time, getting stuck in such local minima happens in the later stages of runs, when the cooling values are quite low, and therefore the probability of moving to a rescuable neighbour approaches 0. In order to avoid E_0, it is required to relax the restricted conditions to let the algorithm proceed by jumping to a solution state that avoids E_0. A multi-start SA algorithm is more useful to relax these conditions rather than a single run SA since the random moves are more uniformly distributed in the multi-start one and it

gives more opportunities to commence new hill climbing cycles in the later stages. Thus, a compact SA algorithm that constantly picks the same solution and manipulates it along a number of iterations for several times can easily avoid the local minima, as it adopts a set of short Markov chains instead of a single and long one. This allows changing the direction of solution path towards a much more useful destination.

The other property of ESA is to tackle a population of solutions rather than an individual. This decreases the effects of initial solutions on the optimisation process. Many works on solving combinatorial optimisation problems by heuristics focused on the effects of initial solutions. When an initial solution has been chosen, there arise limited possible paths to proceed under the certain circumstances since the optimisation process behaves as a Markov chain and each chain offers limited paths to the destination, as widely shown in the literature (Van Laarhoven, Aarts, and Lenstra, 1992; Kolonko, 1999; Steinhofel, Albrecht, and Wong, 1999). Looking at the initial conditions, one can estimate the probability of getting an optimal or useful near optimal solution with a particular initial solution. In fact, all initial conditions are not that easy to avoid the local optima in reasonable time. Therefore, it is not wrong to say that a diverse population of initial solutions gives higher probability than a single initial solution to catch the optimum or a useful near optimum within a reasonable time. Moreover, if useful selection and replacement strategies can be utilised, it will definitely help the process to improve the quality of solutions. So, for that reason, the ESA algorithm is run on a population of solutions rather than an individual using various selection and replacement strategies.

3. Distributed evolutionary simulated annealing (dESA)

As discussed in the previous section, ESA gives new opportunities to commence new valuable hill climbing processes in which the particular solution under consideration may take new opportunities to move to a neighbour that avoids the local minima. Besides, it is preferred to work with bigger populations of solutions for the sake of diversification in evolutionary algorithms. Therefore, the more time given to ESA for manipulating a particular solution, the easier to reach the global optimum. However, this is very time consuming option that forces one to seek alternatives. Since we do not want to lose the benefits of working with larger populations there arises a contradiction between the number of generations or evaluations and the population size used that needs to be resolved.

Suppose we have a population P and a number of generations or evaluations, N and also let the number being selected for ith solution be c_i. The idea is to keep $c_i \cong |P|/N$ so as to give opportunities to every individual to be manipulated enough. On the other hand, if c_i, lasts t_i amount of CPU time, the number of selections for the whole population (c_p) and overall CPU time (t_p) will be:

$$c_p = \sum_{i=1}^{|P|} c_i \quad \text{and} \quad t_p = \sum_{i=1}^{|P|} t_i$$

where $|P|$ is the population size. The contradiction between the population size, which provides the diversity, and the time required should be resolved by using a small-sized

population. Unlike genetic algorithms, we need to work on better-designed small-sized populations to hold the advantages of both the diversity of population and having more selection to be operated by SA. However, it is difficult to have a good spectrum of diversity in small sized populations. As mentioned before, one possible way to overcome this drawback is the consideration of parallel and distributed computing opportunities. The idea is to parallelise the whole system by distributing a larger population over the distributed resources. In order to achieve this, we need to create identical agents that are run on different resources with a sub part of the distributed population.

As is well known and has been mentioned before, there are two main ways to implement a system by parallel computation. The first way, by using physical parallelism, is to partition the whole data set into subparts and distribute them over a running parallel and/or distributed machine or number of processors. Most of the parallel evolutionary systems work in this way. The second way, in which the parallelisation is done on the algorithm itself, rather than partitioning the data, is more complicated. This could be called algorithmic parallelism, and is usually developed for parallel machines such as SIMD and MIMD. Since it is difficult to parallelise an ordinary SA in the sense of algorithmic parallelism, we chose to parallelise the system in the sense of physical parallelism.

3.1. Distributing ESA through multi-agent (island) systems

In order to run dESA, a parallel computing environment is required. Since it is not common to have specific parallel machines such as massively parallel processors etc. easily available, we need to consider the most accessible parallel and distributed environment, which is the common wide/local area network of computers (WAN, LAN). Distributed Resource Machine (DRM) is an infrastructure that provides a distributed problem-solving environment based on mobile agents. It is the distributed infrastructure of DREAM[2] software (Jelasity, Preuβ, and Peachter, 2002), which was developed to solve problems through distributed evolutionary algorithms spread over a massive network of nodes on the Internet. The main aim of this system is to solve the problems based on multi agent systems, which run evolutionary algorithms. The system consists of a scalable network of resources, which works as a peer-to-peer network of nodes spread on physically distributed computers. Each node has incomplete knowledge about the rest of the network and works as the container of all the agents running on a particular computer. The environment has very good functionalities to develop applications such that the agents have good communication and limited mobility. See Paechter et al. (2000) for more information on DRM.

The way of distributing the evolutionary processes over the resources throughout DREAM is to implement the island model. Islands are designed and furnished with various properties, data and algorithms, and then distributed over the DRM network. The DRM environment is developed based on multi-thread programming in Java. It runs the islands as multiple independent running (MIR) technology and provides it with a message passing system (MPS) using connectionist sockets based on TCP/IP protocols. It is required to partition the problem into subparts to be solved through multi-island models. Since ESA has an evolutionary nature and is inherently parallel, we easily used DRM software to develop our distributed ESA (dESA) as a model of multiple island[3] applications. Each island is furnished

with an ESA algorithm identical to the others and a small sub-part of the main population, where the ESA algorithm evolves that population towards an optimum state. Apart from these, it is required to specify the selection rule, the replacement rule, and a migration strategy for the ESA tailored for each island. The islands will evolve their sub-populations with their own ESA and let some individuals migrate to other islands in a specified fashion.

By means of parallelisation, it is more likely to have a shortened overall CPU time (t_p) with the same number of selections (c_p). That is the number of a particular individual selected throughout the whole of the evolutionary process. The CPU time needed for the whole evolutionary process will be shortened linearly in proportion to the number of agents, a. If we use the same notations as before, P could be partitioned into a parts with $P_1, P_2, \ldots, P_a$, denoting the sub-populations enumerated as $1, \ldots, a$, respectively. Assigning each of these sub-populations to each particular island, the number of selections across the whole system will roughly remain the same. The parameters in this case are:

$$c_p = \sum_{j=1}^{a} c_j, \quad c_j = \sum_{i=1}^{|P_j|} c_i, \quad \text{and} \quad t_j = \sum_{i=1}^{|P_j|} t_i$$

where c_j and t_j are the total numbers of selections and the CPU times for jth island. Hence, new $c_p \approx axc_j$ and $t_p = \max\{t_1, t_2, \ldots, t_a\}$. Although c_p is still the same but t_p is much shorter even though some additional time is incurred through the communication among islands. It is clear to say that because the agents run in parallel, concurrency cuts the time required to reach the global optimum. While t_p was calculated by a summation function in non-distributed ESA cases, it is now determined as the maximum of the CPU times spent by each island.

3.1.1. Designing a multi island system. The idea is to partition the main population into several small-sized subparts and assign each to a peer island; each furnished with an identical ESA. For a particular application, a main population (P) is totally iterated for a certain number of iterations (N). Each individual in the population is highly likely to be manipulated for more or less c_j times. In order to keep the likelihood at that level for the individuals, we need to distribute that P-sized population over a number of islands (a) equally, and identify the number of iterations accordingly. If we have designed the system as a group of a islands each will evolve a $|P|/a$-sized population through N/a iterations. One of the islands is the Root Island which in charge of collecting and providing the relevant data such as the initial populations and the "best results" of particular periods. The idea is presented in figure 3, where the islands are fully connected to each other. The dashed arrows represent the connection between any two islands while the straight arrows represent the connection between the root island and any other island. The root island is always located at the machine where the system launched, while the other islands run on any machine connected to the network. As the system is provided with scalability, it is easy to develop larger scale applications that consist of more islands.

3.1.2. Island's specifications. Each island employed in a particular application should be identical to each of the others except that the Root Island has some more duties for

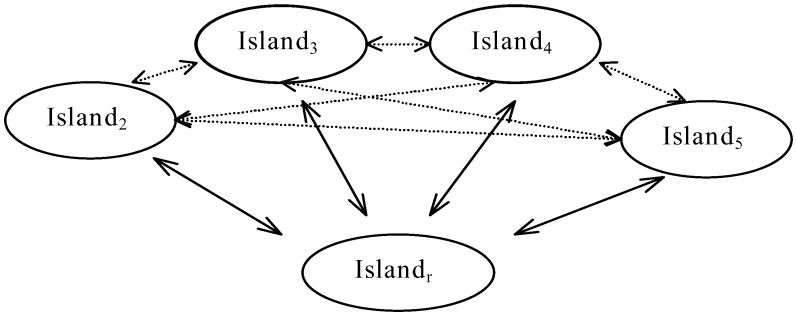

Figure 3. Inter-islands relationships for a dESA application.

administration. The islands are designed so as to evolve the population held towards an optimal solution by using an ESA algorithm. The population is controlled by a specified replacement rule in which the new solutions are promoted. The solution tackled per iteration is selected in a pre-defined way such as tournament, roulette-wheel etc. One selected solution attempts to migrate to another randomly determined island after a predefined period using the Message Passing and Handling System provided by the DRM software. That is used for any communication among the islands as well. This process is repeated throughout N/a iterations, where N and a are the total number of generations/evaluations and the number of island, respectively. The islands have very limited mobility provided by DRM. That is to allow the user to put the islands on any selected node at any time. Whenever an island moves to another node it keeps its-own population and other parameters. Nevertheless, the islands are assigned to the nodes during the launching of the experiments.

The islands communicate with one another by letting the solutions migrate from one to another as well as to report their "best results" to the Root Island at the end of every period. Migration is carried out in such a way that one island randomly selects an individual and sends it to another randomly selected island per predefined period. Whenever an island receives an immigrant, it assesses it according to the replacement rule and makes a decision as to whether to replace it with an individual of its own population or not. Similarly, each island sends its best result to the Root Island per pre-defined period and the Root Island compares all new solutions received and then takes the overall best of the system.

The experiments are launched on DRM by a Launcher agent that creates the Root Island first on the root node. The rest of the islands are also created by the Launcher, providing each of them with the address of the Root Island. The idea is to keep the islands in coordination where the Root Island performs these duties. The Launcher provides the islands with completely different populations; each contains P/a individuals of a desired diversity.

4. Application I: The job shop scheduling (JSS) problems

JSS problems have been worked on for a long time. It is usually difficult to reach the optimal solution in a short time because each problem has a very wide solution space and there is no

guarantee for moving to a better state after reaching any feasible state. Since JSS problems are considered to be among the most difficult problem types of combinatorial optimisation, we aim to solve these problems with dESA algorithm.

4.1. The JSS problems

We are given a set of jobs (J) to be processed in a set of machines (M) subject to a number of technological constraints. Each job consists of m operations, O_j, which must be processed on a particular machine, and each job visits each machine once. There is a predefined order of the operations within a job. Because of this order, each operation has to be processed after its predecessor (PJ_j) and before its successor (SJ_j). Each machine processes n operations in an order that is determined during the scheduling time, although there is no such order initially. Therefore, each operation processed on the M_i has a predecessor (PM_i) and a successor (SM_i). A machine can process only one operation at a time. There are no set-up times, no release dates and no due dates.

Each operation has a processing time (p_{ij}) on related machine starting on r_{ij}. The completion time of o_{ij} is therefore: $c_{ij} = r_{ij} + p_{ij}$ where $i = (1, \ldots, m)$, $j = (1, \ldots, n)$ and $r_{ij} = \max(c_{iPJ_j}, c_{PM_i j})$. Each machine and job have particular completion times, which are identified as: $C_{M_i} = c_{in}$ and $C_{J_j} = c_{jm}$ where c_{in} and c_{jm} are the completion time of the last (nth) operation on ith machine and the completion time of the last (mth) operation of jth job, respectively. The overall objective is to minimise the completion time of the whole schedule (makespan), which is the maximum of machine's completion times, $C_{\max} = \max(C_{M_1}, \ldots, C_{M_m})$. The representation is done via a disjunctive graph, as it is widely used.

4.2. A dESA implementation for JSS problems

We have implemented dESA for solving job shop scheduling problems in the following way. The evolutionary part of the algorithm is configured with a particular set of parameters and methods: the size of population, numbers of evaluations, selection and replacement (evaluation) methods. The application consists of 5 islands; each has a 10-sized population for 2 million iterations, which makes the main population of the system 50 individuals for 10 million iterations. Since this is not a generational evolutionary algorithm, we preferred to count the numbers of evaluations across the whole system, where each island counts each operation by SA as 200 evaluations, as there are 200 evaluations per run of SA operator. Individuals are allowed to migrate to any randomly island per 100,000 evaluations. The migration is subject to the replacement rule, which is applied to any chosen solution.

Our implementation of dESA needs selection and replacement rules as well. Selection of any individual in the population is done uniformly randomly. That is to allow each individual to be selected with equal probability. There are plenty of well-known replacement rules to be applied, but we have implemented a SA like stochastic rule to promote the solutions, which is inspired by Wang and Zheng (2001). The authors have examined three more replacement rules (generational replacement, hill climbing and non-hill climbing) in another work (Aydin and Fogarty, 2002), where a SA like rule outperforms the others. The undertaken rule here

is working in such a way that a SA process is set up according to the number of evaluations. The new solution is replaced with the old if the new is better than the old, otherwise the Boltzman function is running and making the decision of whether to replace or not. The cooling function works after each evaluation to decrease the temperature that is set to control the evaluation.

The SA operator is as described before. The implementation needs a neighbourhood function to permute the tackled solution. In the case of JSS problems, it is required to repair the solutions permuted to have a feasible solution. For this purpose, a semi-active scheduler is developed to do all schedule generation and repairs. The neighbourhood function employed in this implementation is the one based on the longest path of the disjunctive graph, reversing the successive operations on the same machine. This function is inspired by the work of Van Laarhoven, Aarts, and Lenstra (1992), as they have very successfully used it and the fact that the transition between any two critical operators is useful for changing the makespan of a schedule. The change on any other priorities does not cause any cost on the makespan. More information on the efficiency of this function can be seen in Van Laarhoven, Aarts, and Lenstra (1992).

4.3. Experimental results and discussion

In order to illustrate the efficiency of dESA, we have carried out a series of experiments for job shop scheduling problems. The problems tackled are very well known difficult benchmarks, which have been undertaken by various researchers to show the goodness of their methods. First, we developed a sequential (non-distributed) ESA implementation and then a 5-island distributed ESA (dESA) implementation for the same mentioned benchmarks. The general considerations for both cases are as follows:

- the highest temperature is set to 100 to be decayed by 0.955 at each step until 0.01 level of temperature (with this set of parameter levels, one cycle of SA operator takes 200 iterations)
- the algorithm has performed a total of 10 million moves at the end (each ESA operation, a single run of the operator, lasts 200 iterations),
- the size of the population is 50 (which means that each individual has roughly been operated on 1000 times),
- a solution is selected uniformly randomly,
- the new results are replaced only if they satisfy a simulated annealing like stochastic rule.

Both ESA and dESA implementations are set according to parameter levels given above. Since ESA works sequentially and on a particular computer, the population remains 50 and the algorithm works for 10 million evaluations in total. On the other hand, the implementation of dESA in this case is a 5-island implementation in which each island runs to evolve a population of 10 individuals for 2 million iterations. Thus, each individual in this case has got approximately the same number of manipulations as the individuals in the case of the sequential ESA. Since the ESA operator iterates an individual 200 times per cycle, each individual has been selected and operated 1000 times, approximately. The

Table 1. The empirical results obtained by both ESA and dESA implementations.

Problem		ESA				dESA			
Name	Optimum	Mean	SD	Best	Worst	Mean	SD	Best	Worst
ABZ7	655	675.5	2.12	673	678	675.2	2.8	672	678
ABZ8	638	686.8	4.95	683	692	687.2	4.9	681	692
ABZ9	656	699	1.41	698	700	703.0	2.9	699	706
LA21	*1046	1049.4	2.88	1046	1051	1048.4	2.7	1046	1053
LA24	*935	939.2	2.68	935	941	939.6	1.5	938	941
LA25	*977	978.4	2.19	977	982	977.8	1.8	977	981
LA27	*1235	1244.4	4.56	1238	1250	1245.4	3.9	1240	1250
LA29	1130	1177.4	7.54	1173	1188	1182.6	6.1	1176	1190
LA38	*1196	1201.8	3.77	1196	1204	1204.6	3.0	1201	1209
LA40	*1222	1230.8	3.03	1228	1235	1229.2	2.68	1228	1234

ESA works with a population of 50 individuals for 10 million evaluations, while dESA works with a population of 50 individuals distributed over 5 islands equally for 10 million evaluations, one out of five of them (20%) performed by each island.

migration is set up to a period, which is 100000, with a low probability. That means that every 100000-iteration, a randomly selected individual attempts to migrate to another randomly determined island if the random number generated is greater then 0.50. Therefore, every island allows a total of 10 individuals to migrate to the other islands, and accepts in the same number. Experiments have been repeated 50 times per problem.

In Table 1, the results of both ESA and dESA implementations are shown with the mean value, standard deviations, best and worst values found. The optimal and/or lower bound of each problem has been given in the second column adjacent to the problem names, where the values given with asterisks (*) are for the optimum and the others are lower bounds. That means the optimum of benchmarks not indicated by * are still not known. The quality of solutions found by both algorithms is either the highest as hit the optimum or very close to the optimum. Comparing the results, we can evidently see that there is statistically no difference between both implementations with respect to the quality of the solution. The mean, best and worst values are very close to each other. That is because our aim was to get the same quality of solution that we got by ESA within much shorter time by dESA. In fact, the times taken are very different as can be seen on figure 4. The time taken by ESA is given with the upper scattered line and the one by dESA with the lower one. The CPU time level is given on the vertical axes while the benchmarks are indicated on the horizontal axes. All of the CPU times taken by dESA are under 2000 seconds while the time levels taken by ESA differ benchmark by benchmark. The lowest time taken is still about 4800 seconds for benchmarks of LA21, LA23 and LA25. That is because of the size of these benchmarks and their hardness as these 3 are not as hard as the other 7. Overall, the sequential case has taken almost 4 times longer then the distributed one. This proves the significant efficiency of dESA over its sequential version (ESA).

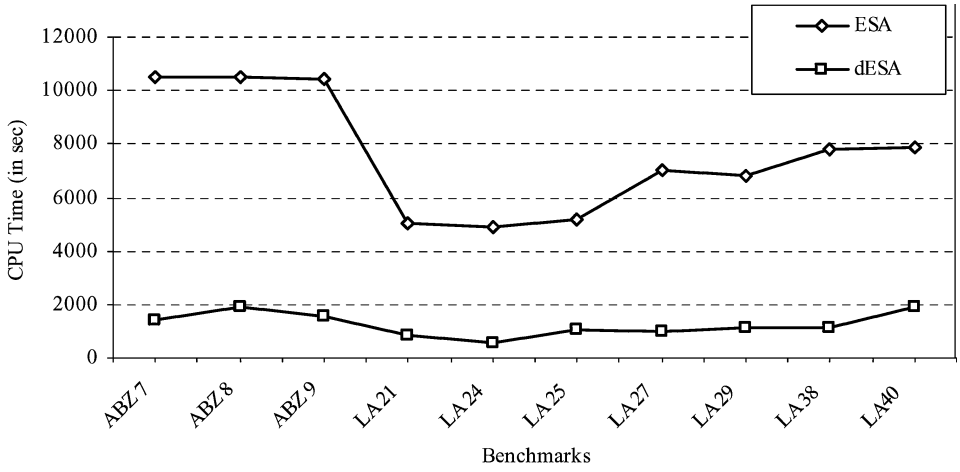

Figure 4. The change of CPU time for each benchmark by both sequential and distributed versions of ESA.

It is not wrong to say that the local minima problems can be overcome by diversification of the solutions. The worse results we got are most likely caused by the low diversity of the population. We suspected that, if we increase the diversity of populations of the islands by changing the rate of migration, the inter-population effect might make our method much better. In fact, we applied two more strategies for migration besides the first one: one does not allow any migration at all, the other works in such a way that every 100000-iteration the island allows an individual to migrate with migration rate of 0.75. That changes the number of migrated individuals from 10 to 15 per island. The results for all three cases are shown in figure 5 presenting error % between the average of values found and optimal ones on the vertical axis. On the horizontal axis, the benchmarks are given as ABZ7, ABZ8, ABZ9, LA21, LA24, LA25, LA27, LA29, LA38 and LA40, respectively. As is seen, there is no significant difference among those three strategies. It helps very little to keep the populations of islands diverse. Migration does not work as it is expected since the algorithm running by each island is exactly the same. If one dispatches various algorithms over islands, then it will work and help to converge faster.

4.4. Related works and conclusions

There are enormous numbers of studies carried out on classical job-shop scheduling problems. We have filtered the most relevant ones in terms of publication date, the problems tackled, and the methods employed to make comparison with our results where SA, GA and their combinations have been considered. In fact, we have not come across any parallel method that solved job shop scheduling problems using the benchmarks undertaken. Only Steinhofel, Albrecht, and Wong (2002) discusses some parallel heuristics for simulated annealing-based algorithms to be applied to job shop scheduling problems. They theoretically criticise the calculation of longest path of schedules by clarifying the dependency on

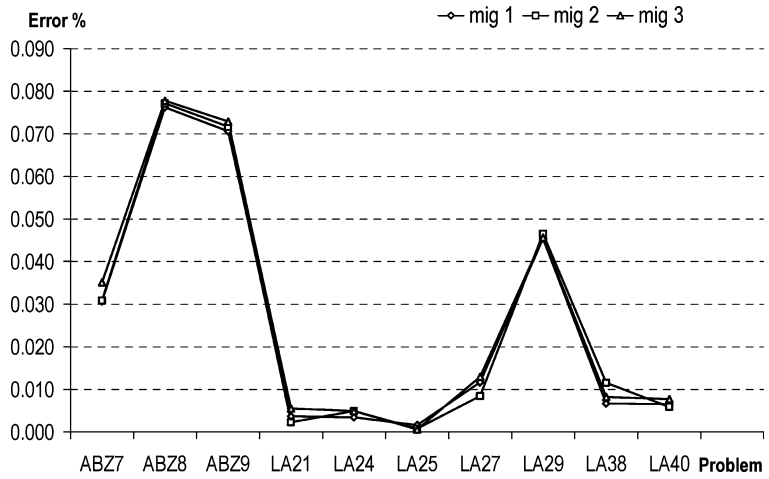

Figure 5. Error percentage of found results respecting to each migration strategy: mig1 lets an individual migrate to another island every 100000-iteration with 0.50 migration rate, while mig2 does not let individual migrate and mig3 lets one migrate per 100000-iteration with 0.75 migration rate.

the number of jobs and machines in an algorithmic parallelism point of view. This is an advanced version of their previous work, which discusses two SA heuristics for job shop scheduling problems (Steinhofel, Albrecht, and Wong, 1999). They presented computational results for very few benchmarks that are worse than ours with respect to the time. The algorithmic parallelism by TCP/IP protocols does not look very useful because the communication time nullifies the time saving advantage. For this reason, we feel that it would not be useful to parallelise the algorithms, as Steinhofel et al. did, by using DRM properties. Wang and Zheng (2001) developed a hybrid optimisation strategy by embedding a SA into GA for job shop scheduling problems. The benchmarks undertaken in their paper are only moderately difficult and our system very quickly finds the optima on them.

Table 2 shows some comparable results from different recent works, each relevant with respect to solving job shop scheduling problems by SA and/or GA; where our results are generally better than the others. None of them are parallel or distributed algorithms. The reason to consider them for comparison is their relevancy to the problems undertaken as well as to give an idea on how close our approach and some other works got to the optimum values. The aim is not to compare their successes with respect to CPU time taken. Satake et al. (1999) present a rescheduling based SA approach with the results given in the second part of Table 2, running the experiments on an IBM-PC Pentium 133 MHz, while KTM used SUN SPARC workstation 2 for a taboo search based approach as reported by Satake et al. (1999). Our results are better than theirs with respect to the length of schedules (the quality of the solution). However, Kolonko (1999) has better results from a combined GA-SA system for the last four benchmarks respecting length of schedules, too. We have run our experiments on a network of nodes, each one lives on a particular machine and the machines are not homogeneously configured. We believe that if the other algorithms such as Kolonko's one were developed in Java and run in the same configuration as we used, the

Table 2. Some experimental results from the literature to be compared with dESA with respect to the quality of the solutions.

Problem		Sakate et al.		KTM		Kolonko		dESA	
Name	Opt	Mean	Best	Mean	Best	Mean	Best	Mean	Best
ABZ7	655	684.8	679	–	–	–	–	675.2	672
ABZ8	638	698.0	684	–	–	–	–	687.2	681
ABZ9	656	706.6	698	–	–	–	–	703.0	699
LA21	*1046	1062.0	1046	1050.0	1047	1051.0	1047	1048.4	1046
LA24	*935	949.0	936	942.0	941	940.4	938	939.6	938
LA25	*977	989.6	980	980.5	979	979.0	977	977.8	977
LA27	*1235	1263.2	1248	1247.3	1241	1244.8	1236	1245.4	1240
LA29	1130	1196.4	1167	1173.3	1165	1169.2	1167	1182.6	1176
LA38	*1196	1218.4	1202	1210.5	1202	1202.4	1201	1204.6	1201
LA40	*1222	1243.0	1233	1233.7	1228	1228.6	1226	1229.2	1228

superiority of our algorithm would be more evident in terms of CPU times as well. Another significant benefit of our method is its scalability which has been discussed in Section 5.4. For instance, if we partition a bigger population over 10 islands rather than 5 islands for the same number of iterations per individual, the CPU time will be much better, as we will be increasing the diversity of the population.

5. Application II: Uncapacitated facility location problems

Facility Location Problems that have a NP-Hard nature have been playing an important role in operation research and manufacturing context. The uncapacitated facility location (UFL) problem, also known as simple plant location problem, is basically a member of the family of location problems. In a general form, the problem is to determine the optimal number of facility echelons, the number and location of facilities in each facility echelon, the assignment of distribution activities/commodities to facilities, and the allocation of customer demand among the facilities. The UFL problems have several applications, such as the bank account location problem, the clustering problem, economic lot sizing, vehicle routing, network design, distributed data and communication networks.

UFL problems have been studied for many years and thus there is a very rich literature in operations research (OR) for this kind of problem. Since they are NP-Hard problems, the larger the size of the problem, the harder to find the optimal solution and furthermore, the longer to reach reasonable results. During the past three decades, these UFL problems have been triggered and examined extensively (Kratica et al., 2001) by various attempts and approaches. The presentation of all-important contributions relevant to UFL problems can be summarised as follows. There are two main categories of approach to this type of problem:

classical OR, (branch and bound, primal and dual ascent methods, linear programming and Lagrangean relaxation algorithms) and meta-heuristic based methods. The Dualoc algorithm (Erlenkotter, 1978) is one of the most respected methods based on OR approaches as the fastest one for UFL problems for a long time. It is based on a linear programming dual formation (LP dual) in condensed form that evolved in simple ascent and adjustment procedures. If ascent and adjustment procedures do not find the optimal solution, Branch-and-Bound (BnB) procedure completes the solution process.

Guignard (1985) proposed to strengthen the separable Lagrangean relaxation of the UFL problems by using Bender's inequalities generated during a Lagrangean dual ascent procedure. The coupling of that technique with a good primal heuristic could reduce the integrity gap. Simao and Thizy (1989) presented a streamlined dual simplex algorithm designed on the basis of a covering formulation of the UFL problem. Their computational experience with standard data sets indicates the superiority of dual approaches. Koerkel (1989) showed how to modify a primal-dual version of Erlenkotter's (1978) exact algorithm to get an improved procedure. The computational experience with large-scale problem instances indicated that speedup to Dualoc is significant (more than one order of magnitude). Coon and Cornuejols (1990) present a method based upon the exact solution of the condensed dual of LP relaxation via orthogonal projections. In Holmberg (1995) and Holmberg and Jornsten (1996) a primal-dual solution approach based on decomposition principles is used. They fixed some variables in the primal sub-problem and relaxed some constraints in the dual sub-problem. By fixing their Lagrange multipliers, both of these problems become easier to solve than the original one. The computational tests proved the advantageous in comparison to the dual ascent method of Erlenkotter.

Kratica et al. (2001) have applied genetic algorithms to UFL problems to solve 1000×1000-sized customer-facility instances. They considered many benchmarks within the literature to be solved by their algorithm in addition to their own similar large size problems. They compared their results with Erlenkotter's dual-based algorithm (DUALOC) showing that their algorithm is much more efficient than DUALOC for problems larger than 100×1000. Although DUALOC has better results for some benchmarks, it is worse in time consumption. Jaramillo, Bhadury, and Batta (2002) applied a genetic algorithm that is mainly based on the operators that Beasley and Chu (1996) applied in their genetic algorithms for covering problems. They compared their results with a Lagrangean relaxation algorithm presented by Beasley (1993). Although Kratica et al. (2001) have proposed a very similar approach, Jaramillo, Bhadury, and Batta (2002) give a fresher and less time consuming approach with which it is much fairer to make a comparison.

As is mentioned in the literature, simulated annealing approaches may be very successful in terms of the quality of the solutions but not so impressive with respect to the CPU times. This is not the first SA approach to solve UFL problems. Alves and Almeida (1992) have solved UFL problems but taking far longer time. The aim of this application is to show that the dESA implementations can get the best quality of solutions within shorter times. We have examined some sequential simulated annealing approaches and one of the recent genetic algorithm approaches (Jaramillo, Bhadury, and Batta, 2002) for UFL problems for some useful comparisons.

5.1. The UFL problem

The mathematical formulation of these problems as mixed integer programming models has proven very fruitful in the derivation of solution methods. To formalise the UFL problems, consider a set of candidate sites for facility location, $J = \{1, \ldots, m\}$, and a set of customers, $I = \{1, \ldots, n\}$. Each facility $j \in J$ has a fixed cost, f_j. Every customer $i \in I$ has a demand, b_i, and c_{ij} is the unit transportation cost from facility j to customer i. Without a loss of generality we can normalise the customer demands to $b_i = 1$. The problem is formulated in the following way:

$$Z = \min \sum_{i=1}^{n} \sum_{j=1}^{m} c_{ij} x_{ij} + \sum_{j=1}^{m} f_j y_j \tag{1}$$

subject to:

$$\sum_{j=1}^{m} x_{ij} = 1, \quad \text{for } \forall\, i \in I; \tag{2}$$

$$0 \leq x_{ij} \leq y_j \quad \text{or} \quad y_j \in \{0; 1\}; \quad \text{for } \forall\, i \in I \quad \text{and} \quad \forall\, j \in J; \tag{3}$$

where x_{ij} represents the quantity supplied from facility j to customer i, y_j indicates whether facility j is established ($y_j = 1$) or not ($y_j = 0$). The constraint (2) makes sure that all demands have been met by the open sites, and the constraint (3) is to keep integerity. Since it is assumed that there is no capacity limit for any facility, the demand size of each customer is ignored, and therefore constraint (2) established without considering demand variable ($b_i = 1$). By this model, it is mainly decided for the number of facility sites to be established. It could be possible to determine the quantities to be supplied from facility j to customer i such that the total cost (including fixed and variable costs) is minimised. However, it is not considered in this case since each candidate site is assumed to have unlimited capacity.

5.2. A dESA implementation for UFL problems

The dESA implementation for UFL problems is a little bit different from the implementation for JSS problems. The main difference is the size of sub populations. In JSS case, each island has a sub-population to evolve, on the other hand, the islands take individuals instead of sub populations in the case of UFL. That is because the problems are not as hard as JSS benchmarks, and are very time sensitive due to their sizes. Since it takes longer to operate an individual once by ESA than any other genetic operator, it is crucial to balance the size of population for ESA. The selection and replacement rules remain the same as in the JSS application, but there is no migration in this case. More relevant details come in the following section.

UFL problems are usually represented in a binary way in which the open facilities are denoted by 1 and the closed ones by 0. In order to move from one solution to another, the

digits are briefly converted to each other. Depending on the heuristics employed, this conversion is done either by a neighbourhood function, as happens here, or a genetic operation. We represent the state of solutions by a list of integers, say L, denoting the enumerated open facility, (l_i). For instance, $L = \{0, 1, 4, 12\}$ means the open facilities are Number 0, 1, 4 and 12, the rest are considered as closed. As we denote the particular solution state at time t with s_t, one state of solution will be: $s_t = \{L\} = \{l_i \mid 0 < i \leq m\}$ where m is the maximum number of facilities.

The neighbourhood function employed allows modifying the states by three different operations: (1) exchanges one integer on the list with another possible one; (2) adds a new integer to the list; (3) removes one integer from the list. That means that (1) closes an open facility and opens another, (2) opens a new facility and (3) closes an open one. Only one of these operations is applied at once. We select one operation randomly according to the following rule:

$$\text{operator} \longleftarrow \begin{cases} \text{Exchange( )} & (\lambda(L) = 1 \cap 0 \leq \rho < 0.7) \cup (\lambda(L) > 1 \cap 0 \leq \rho \leq 0.5) \\ \text{Add( )} & (\lambda(L) = 1 \cap 0.7 \leq \rho < 1) \cup (\lambda(L) > 1 \cap 0.5 \leq \rho \leq 0.7) \\ \textit{Remove( )} & (\lambda(L) = m) \cup (\lambda(L) < m \cap 0.7 \leq \rho \leq 1) \end{cases}$$

where $\lambda(L)$ is the length of the list, ρ is a uniformly generated random number, and m is the maximum number of facilities as usual. By applying this function, we move to a neighbouring state. This is a preventive neighbourhood function that keeps the solutions feasible by letting the operators manipulate as is convenient. The convenience of one situation is determined by both the length of the list and the random number $(\lambda(L), \rho)$. For instance, if $\lambda(L) > 1$ then any of the operations can be selected according to ρ, on the other hand, if $\lambda(L) = m$ then only *Remove( )* operator is allowed to operate.

5.3. Experimental results and discussion

The experimental study for this application is carried out as a comparative work. First, a genetic algorithm (GA) introduced by Jaramillo et al. (2002) and then two sequential (non-distributed) implementations of ESA have been developed. Finally, dESA has been implemented as described before. All ESA implementations including dESA have 100 of highest temperature to be decayed by 0.955 at each cooling iteration until the temperature becomes 0.01. That takes 200 iterations per run of the SA operator. This experimental study for sequential algorithms has been done on a Pentium III 700 MHz double processor computer, running Windows 2000. All of the software was developed using Sun Java JDK1.3.1. The GA introduced was encoded in Java in order to make the comparison fairer. In fact, it is reported that it has been encoded in FORTRAN 90 and executed on PC equipped by Intel Pentium running at 200 MHz. Regarding the big difference between our conditions and theirs, it would not be fair, if we just considered their results as standing in the paper. We encoded the algorithm in Java without missing any details given in the paper. Two ESA implementations are different from each other just with respect to the size of population and the number of iterations. The tackled problems are very well known benchmarks that are accessible on OR Library (Beasley, 1996). The dESA implementation is developed as

Table 3. Benchmarks tackled with the sizes and the optimum values.

Benchmark	Size ($n \times m$)	Optimum
Cap 71	16 × 50	932615.75
Cap 72	16 × 50	977799.40
Cap 73	16 × 50	1010641.45
Cap 74	16 × 50	1034976.98
Cap 101	25 × 50	796648.44
Cap 102	25 × 50	854704.20
Cap 103	25 × 50	893782.11
Cap 104	25 × 50	928941.75
Cap 131	50 × 50	793439.56
Cap 132	50 × 50	851495.33
Cap 133	50 × 50	893076.71
Cap 134	50 × 50	928941.75
Cap A	100 × 1000	17156454.48
Cap B	100 × 1000	12979071.58
Cap C	100 × 1000	11505594.33

a 12-asynchronous-island model; each evolves a single individual for a while. More details are coming in the following paragraphs. The benchmarks are introduced in Table 3 with their sizes and the optimum values.

The first simulated annealing application, which is called ESA-I, has been set up with a population of 5 individuals to be evaluated by SA operator for 300 times so that the total number of evaluation becomes 60000 (300 × 200), since the SA operator evaluates an individual for 200 times per run. Therefore, the evaluation per individual is done approximately 12000 times. On the other hand, the second simulated annealing application, which is called ESA-II, has been developed with a population of 10 individuals to be evolved for 2000 iterations so as the total number of evaluations becomes 400000 and the evaluations per individual become approximately 40000. As mentioned before, the dESA implementation consists of 12 identical ESA islands, where each works independently and communicates autonomously, tackling a single individual rather than a population. The idea here is to spread a population of solutions over the distributed islands to have alternative operations in parallel.

The results of all four applications are summarised in Table 4 showing the superiority of the dESA over the others with respect to both the quality of the solution and CPU time taken. The results are shown in the table of two groups of columns: one is for the quality of solutions the other is for the CPU time spent. The quality of solutions is given as a percentage that is calculated as follows:

$$Difference\% = \frac{Result - Optimum}{Optimum}$$

Table 4. Summary of results gained from different algorithms for comparison.

	Difference%				CPU Time in sec			
Benchmarks	GA	ESA-I	ESA-II	dESA	GA	ESA-I	ESA-II	dESA
Cap 71	0.0	0.0	0.0	0.0	0.287	0.041	0.040	0.013
Cap 72	0.0	0.0	0.0	0.0	0.322	0.028	0.020	0.014
Cap 73	0.00033	0.0	0.0	0.0	0.773	0.031	0.022	0.011
Cap 74	0.0	0.0	0.0	0.0	0.200	0.018	0.013	0.008
Cap 101	0.00020	0.0	0.0	0.0	0.801	0.256	0.250	0.046
Cap 102	0.0	0.0	0.0	0.0	0.896	0.098	0.085	0.037
Cap 103	0.00015	0.0	0.0	0.0	1.371	0.119	0.156	0.115
Cap 104	0.0	0.0	0.0	0.0	0.514	0.026	0.031	0.010
Cap 131	0.00065	0.00008	0.0	0.0	6.663	2.506	1.960	0.207
Cap 132	0.0	0.0	0.0	0.0	5.274	0.446	0.828	0.160
Cap 133	0.00037	0.00002	0.0	0.0	7.189	0.443	0.991	0.103
Cap 134	0.0	0.0	0.0	0.0	2.573	0.079	0.111	0.019
Cap A	0.0	0.0	0.0	0.0	184.422	17.930	29.699	1.392
Cap B	0.00172	0.00070	0.0	0.0	510.445	91.937	98.563	7.995
Cap C	0.00131	0.00119	0.00011	0.0	591.516	131.345	184.263	18.017

The quality of results found by GA are slightly worse than those already given by authors in the paper, this may be because of the random numbers used in the execution. The most important aspect is the time consumed for finding these solutions, which are much longer than those that are given in the paper. The reason behind this may be the time Java takes compared with the other languages. The results of ESA-I hit the optimum values in 11 of 15 benchmarks, but have deviations in 4 of 15. They are better than those found by GA as GA hit 8 of 15 optimum values. We realised that the four benchmarks that ESA-I has deviations from the optima are really tougher than the others. Experiments are done 50 times for each benchmark. The CPU time consumed is measured as the time of the last best result found. The results of ESA-II are much better in terms of quality of solutions, but not that much better in terms of the CPU time even though it is almost the same in some benchmarks. But, still benchmark Cap C is not met for 100% as it is not met the optimum in few runs out of 50. A controversial situation arises comparing the results of both ESA algorithms. ESA-I is slightly better than ESA-II with respect to CPU time, but slightly worse in terms of quality of the results. In fact, ESA-I has a lower size of population and number of iterations than ESA-II. This shows that the larger the size of population, the longer the time needed to process. Thus, it is necessary to let the systems consume more time when more precise results are desired.

The results of dESA are very impressive and the best of all the algorithms examined. All benchmarks have been solved reaching the optimal values in a very short time compared with the others. The main difficulty with the methods working with individuals is the more likely trap of sticking in local optimum depending on the initial solution. In the case of

population based methods, the diversity of solutions lets the algorithm easily go through various paths towards the optimum solution so that it accomplishes the mission of reaching the optimum solution easily. In the case of dESA, a diverse initial population is spread over the islands to run all in parallel, and at the end, the best of solutions are found in terms of both time and quality. This makes the method more powerful than any of the others.

5.4. Parallelism and scalability

One of the main benefits of multi agent systems run on parallel environments is to let the implementations be scalable. By the means of parallelism one can build systems more scalable so as to solve the problems more efficiently. As mentioned in the Section 4.4, scalability is one of the strengths of this algorithm inherited from DRM, as one can enhance the system by increasing the numbers of the islands. The population is getting larger and more diverse and the time spent does not change by the means of parallelism when the number of the islands grows. Figure 6 indicates the decrease in CPU time gained by increasing the number of islands showing. We examined the three hardest benchmarks (Cap A, Cap B, and Cap C) with four different numbers of islands: 5, 10, 15, and 20 for this purpose. On vertical and horizontal axises, the average CPU time and the numbers of the islands are presented, respectively. The time spent for each case is presented on the corresponding data point. The lowest graph presents the change of time for Cap A, while the middle is for Cap B and the top one is for Cap C. As Cap A is not as hard as the other two, there is no significant change in the CPU time. On the other hand, there is a substantial and significant decrease

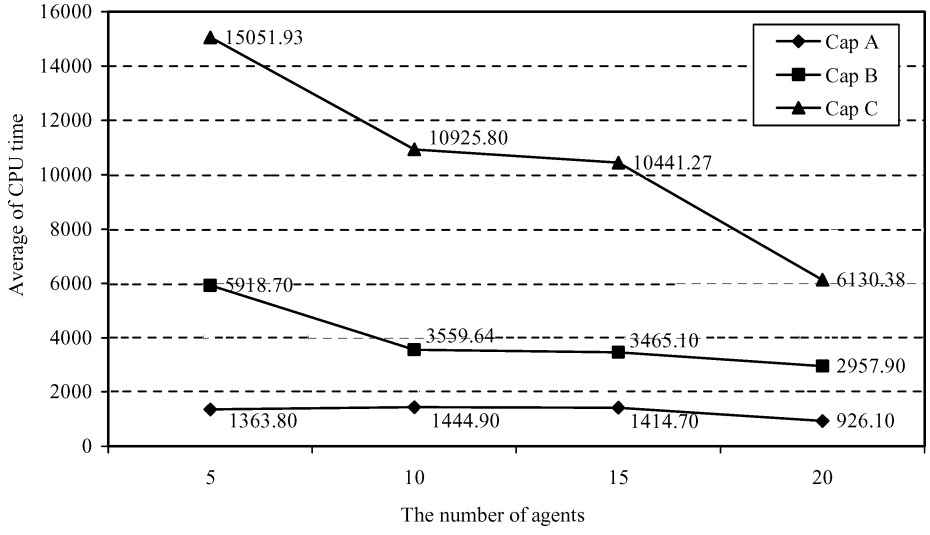

Figure 6. The change in the speed of convergence of dESA by changing the numbers of islands (scalability).

in CPU time by increasing the numbers of islands, as indicated in the case of Cap C and the earlier cases of Cap B. The CPU time is about 15000 seconds when the number of islands is 5 and it gets down to 6000 by employing 20 islands. The same situation is the case for Cap B as the CPU time decreases from 5920 to 3560 seconds by employing 5 more islands. This clearly shows that the larger the numbers of island, the shorter the CPU time spent to reach the optimum. The reason for the slightly change in the case of Cap A and the last examinations of Cap B could be that these are the minimum time needed to get the optimum value by dESA under the circumstances of the configuration held.

6. Conclusions

This paper presents an evolutionary simulated annealing algorithm and its distributed version with two applications in combinatorial optimisation: job shop scheduling and uncapacitated facility location problems. As widely discussed in the literature, simulated annealing algorithms may guarantee the optimum or a useful near optimum result. However, it may not provide them in an affordable time. For this reason, SA is either hybridised with another heuristic such as GA or parallelised to run on multiple resources. dESA is a distributed evolutionary simulated annealing algorithm that runs on multiple agent systems, and is very easily implementable. As WAN/LAN systems are more common multi machine systems, dESA is implemented based on those systems. It is an evolutionary and a distributed algorithm that runs an evolutionary algorithm that consists of a SA operator instead of reproduction and a selection operator. Since it works as a proper evolutionary algorithm and easily avoids the local minima traps, the convergence is faster than an ordinary SA algorithm. It solves problems far faster when it is implemented on distributed systems. It is scalable to speed up the problem solvers and/or to solve extra large size problems. The scalability is shown with experiments in the second application (UFL).

Job shop scheduling is very successful application on which dESA outperforms many recent approaches. One of the next possible step for this research will be to investigate how scalability helps to solve larger JSS problems. In the uncapacitated facility location application, we have not tried larger populations for islands, since the benchmark sizes are not that convenient to do this. One of the future works may be to examine larger UFL problems with bigger multi island systems. Furthermore, larger problems can be decomposed and solved via dESA.

Acknowledgment

This work was funded as part of the European Commission Information Society Technologies Programme (Future and Emerging Technologies). The authors have sole responsibility for this work, it does not represent the opinion of the European Community, and the European Community is not responsible for any use that may be made of the data appearing herein. And also, the authors acknowledge the invaluable discussions with Dr. Vecihi Yigit during his visit to London South Bank University.

Notes

1. We would like to distinguish multi-start with multi-run, as multi-run is used for parallel processing by multi thread programming.
2. http://www.world-wide-dream.org
3. Island is the name of an agent in the DREAM vocabulary, as well.

References

Alba, E. and M. Tomassini. (2002). "Parallelism and Evolutionary Algorithms." *IEEE Transactions on Evolutionary Computation* 6(5), 443–462.

Alves, M.L. and M.T. Almeida. (1992). "Simulated Annealing Algorithm for the Simple Plant Location Problem: A Computational Study." *Rev. Invest*. 12.

Aydin, M.E. and T.C. Fogarty. (2002). "A Modular Simulated Annealing Algorithm for Multi-Agent Systems: A Job Shop Scheduling Application." In *Proc. of ICRM 2002 (2nd International Conference of Responsive Manufacturing)*. 26–18 June, Gaziantep, Turkey.

Beasley, J.E. (1993). "Lagrangean Heuristics for Location Problems." *European Journal of Operational Research* 65, 383–399.

Beasley, J.E. (1996). "Obtaining Test Problems *via* Internet." *Journal of Global Optimisation* 8, 429–433, http://mscmga.ms.ic.ac.uk/info.html.

Beasley, J.E. and P.C. Chu. (1996). "A Genetic Algorithm For the Set Covering Problem." *European Journal of Operational Research* 94, 392–404.

Bevilacqua, A. (2002). "A Methodological Approach to Parallel Simulated Annealing on an SMP System." *Journal of Parallel and Distributed Computing* 62(10), 1548–1570.

Bhandarkar, S.M., S. Machaka, S. Chirravuri, and J. Arlond. " Parallel Computing for Chromosome Reconstruction via Ordering of DNA Sequences." *Parallel Computing* 24(12/13), 1177–1204.

Boissin, N. and J.L. Lutton. (1993). "A Parallel Simulated Annealing Algorithm." *Parallel Computing* 19(8), 859–872.

Bongiovanni, G., P. Crescenzi, and C. Guerra. (1995). "Parallel Simulated Annealing for Shape Detection." *Computer Vision and Image Understanding* 61(1), 60–19.

Chu, K.W., Y. Deng, and J. Reinitz. (1999). "Parallel Simulated Annealing by Mixing of States." *Journal of Computational Physics* 148(2), 646–662.

Conn, A.R. and G. Cornuejols. (1990). "A Projection Method for the Uncapacitated Facility Location Problem." *Math. Programming* 46, 273–298.

Erlenkotter, D. (1978). "A Dual-Based Procedure for Uncapacitated Facility Location." *Operations Research* 26, 992–1009.

Ferreira, A.G. and J. Zerovnik. (1993). "Bounding the Probability of Success of Stochastic Methods for Global Optimisation." *Computers & Mathematics with Applications* 25(10/11), 1–8.

Greening, D. R. (1990). "Parallel Simulated Annealing Techniques." *Physica D: Nonlinear Phenomena* 42(1–3), 293–306.

Guignard, M. (1988). "A Lagrangean Dual Ascent Algorithm for Simple Plant Location Problems." *European Journal of Operational Research* 35, 193–200.

Holmberg, K. (1995). "Experiments with Primal-Dual Decomposition and Subgradient Methods for the Uncapacitated Facility Location Problem." Research Report LiTH-MAT/OPT-WP-1995- 08, Optimization. Department of Mathematics, Linkoping Institute of Technology, Sweden.

Holmberg, K. and K. Jörnsten. (1996). "Dual Search Procedures for The Exact Formulation of The Simple Plant Location Problem with Spatial Interaction." *Location Science* 4, 83–100.

Huang, H.C., J.S. Pan, Z.M. Lu, S.H. Sun, and H.M. Hang. (2001). "Vector Quantization Based on Genetic Simulated Annealing." *Signal Processing* 81, 1513–1523.

Jaramillo, J.H., J. Bhadury, and R. Batta. (2002). "On the Use of Genetic Algorithms to Solve Location Problems." *Computers & Operations Research* 29, 761–779.

Jeong, I.K and J.J. Lee. (1996). "Adaptive Simulated Annealing Genetic Algorithm for System Identification." *Engineering Applications of Artificial Intelligence* 9(5), 523–532.

Jelasity, M., M. Preuβ, and B. Peachter. (2002). "A Scalable and Robust Framework for Distributed Applications." *CEC'02: The 2002 World Congress on Computational Intelligence*, Honolulu, HI, USA.

Kim, Y., Y. Jang, and M. Kim. (1991). "Stepwise-Overlapped Parallel Simulated Annealing and its Application to Floorplan Designs." *Computer-Aided Design* 23(2), 133–144.

Koerkel, M. (1989). "On the Exact Solution of Large-Scale Simple Plant Location Problems." *European Journal of Operational Research* 39, 157–173.

Kirkpatrick, S., C.D. Jr. Gelatt, and M.P. Vecchi. (1938). "Optimisation by Simulated Annealing." *Science* 220(4598), 671–679.

Kolonko, M. (1999). "Some New Results on Simulated Annealing Applied to Job Shop Scheduling Problem." *European Journal of Operational Research* 113, 123–136.

Kratica, J., D. Tošiæ, V. Filipoviæ, and I. Ljubiæ. (2001). "Solving the Simple Plant Location Problem by Genetic Algorithms." *RAIRO—Operations Research* 35(1), 127–142.

Paechter, B., T. Back, M. Schoenauer, M. Sebag, A.E. Eiben, J.J. Merelo, and T.C. Fogarty. (2000). "A Distributed Resource Evolutionary Algorithm Machine (DREAM)." In *Proc. of the Congress of Evolutionary Computation 2000 (CEC2000). IEEE, 2000*, IEEE Press, pp. 951–958.

Satake, T., K. Morikawa, K. Takahashi, and N. Nakamura. (1999). "Simulated Annealing Approach for Minimising the Makespan of the General Job-Shop." *International Journal of Production Economics* 60/61, 515–522.

Schmeck, H., J. Branke, and U. Kohlmorgen. (2001). "Parallel Implementations of Evolutionary Algorithms." In A. Zomaya, F. Ercal, and S. Olariu (eds.), *Solutions to Parallel and Distributed Computing Problems*. John Wiley and Sons Inc.

Simao, H.P. and J.M. Thizy. (1989). "A Dual Simplex Algorithm for the Canonical Representation of the Uncapacitated Facility Location Problem." *Operations Research Letters* 8, 279–286.

Steinhofel, K., A. Albrecht, and C.K. Wong. (1999). "Two Simulated Annealing-Based Heuristics for the Job Shop Scheduling Problem." *European Journal of Operational Research* 118, 524–548.

Steinhofel, K., A. Albrecht, and C.K. Wong. (2002). "Fast Parallel Heuristics for the Job Shop Scheduling Problem." *Computers & Operations Research* 29, 151–169.

Wang, L. and D.Z. Zheng. (2001). "An Effective Hybrid Optimisation Strategy for Job-Shop Scheduling Problems." *Computers & Operations Research* 28, 585–596.

Wong, S.Y.W. (2001). "Hybrid Simulated Annealing/Genetic Algorithm Approach to Short Term Hydro-Thermal Scheduling with Multiple Thermal Plants." *Electrical Power & Energy Systems* 23, 565–575.

Van Laarhoven, P.J.M., E.H. Aarts, and J.K. Lenstra. (1992). "Job Shop Scheduling by Simulated Annealing." *Operations Research* 40(1), 113–125.

Vales-Alonso, J., J. Fernandez, F.J. Gonzalez-Castano, and A. Cabarello. (2003). "A Parallel Optimization Approach for Controlling Allele Diversity in Conversation Schemes." *Mathematical Biosciences* 183(2), 161–173.

Voogd, J.M., P.M.A. Sloot, and R. Dantzig. (1994). "Crystallization on a Shape." *Future Generation Computer Systems* 10(2/3), 359–361.

Yong, L., K. Lishan, and D.J. Evans. (1995). "The Annealing Evolution Algorithm as Function Optimizer." *Parallel Computing* 21(3), 389–400.