Some aspects concerning the implementation of a parallel hybrid metaheuristic

Viorel Minzu*, Liviu Beldiman

Department of Control Systems and Industrial Informatics, “Dunarea de Jos” University of Galati, 47, Domneasca Street, 80008 Galati, Romania

Received 8 January 2005; received in revised form 30 November 2006; accepted 13 December 2006

Available online 12 February 2007

Abstract

This paper presents a discrete optimization system implementing a parallel hybrid metaheuristic. This is obtained by joining a genetic algorithm and a parallel version of a stochastic descent method called ‘Kangaroo’. Two real problems in the manufacturing field were solved using the proposed metaheuristic. This offered the opportunity to underline some aspects regarding the implementation of this hybrid system. The impact of the precedence constraints upon the implementation of the genetic operators (crossover and mutation) is also considered.

Keywords: Metaheuristics; Manufacturing; Genetic algorithm; Discrete optimization; Stochastic descent

1. A parallel hybrid metaheuristic

The majority of the optimization problems in the manufacturing field are unfortunately NP-complete. For example, the single-machine scheduling problem (SMSP) (Baker, 1974) and the tasks-to-workstations assignment (TWA) problem (Minzu and Henrioud, 1998) are optimization problems of this type because of their combinatorial aspect. Hence, sub-optimal solutions are generally preferred to optimal ones.

The iterative methods are, undoubtedly, the most efficient methods for finding sub-optimal solutions for NP-hard problems. Among these methods, two families may be distinguished:

- the iterative methods that improve a unique solution (ISI—iterated solution improvement). Tabu search (Glover, 1989), simulated annealing (Kirkpatrick et al., 1983) and stochastic descent methods (Papadimitriou and Steiglitz, 1982) (like the Kangaroo method) are ISI methods and
- the iterative methods that improve a set (population) of solutions (IPI—iterated population improvement).

Genetic algorithms (Holland, 1975) are an example of the IPI method.

In the last years, hybrid metaheuristics (Vaessens et al., 1992; Taillard et al., 1998; Mahfoud and Goldberg, 1995; Blum and Roli, 2003) were developed, giving very interesting results. For example, a hybrid metaheuristic formed by a GA and a tabu search is very efficient for many optimization problems. This paper proposes a parallel hybrid metaheuristic made up of an IPI and an ISI method that is presented in Fig. 1. The IPI method is a GA and the ISI method is a parallel Kangaroo algorithm (KA). The authors have already proposed in Minzu and Beldiman (2003) this metaheuristic for solving the SMSP. The study presented in that paper was dedicated to this problem and the implementation of the proposed hybrid system was in an early stage. In this paper, an analysis of the general implementation of this metaheuristic is proposed. This analysis is done by studying the implementations of two quite different problems: SMSP and TWA problem, which are at the same time real problems in the manufacturing domain. Moreover, some practical aspects concerning the treatment of the precedence constraints between tasks are discussed. These constraints have to be met in problems like TWA.
A genetic algorithm (GA) performs the first phase of the system and supplies good solutions after a relative small number of iterations. The initial population of the GA is yield by the initial population generation module. The evolution of this population is considered along a constant number of generations, $N_{\text{gen}}$, so that, at the end of this phase, the final population contains solutions placed in different regions of the search space. This GA has two characteristics:

- The value of $N_{\text{gen}}$ is much smaller than the number of iterations of a stand-alone GA aiming to solve the considered problem.
- The probability of applying the mutation operator is greater than the one used in a stand-alone GA. Therefore, the diversifying character is strengthened.

A parallel KA performs the intensification phase. The Kangaroo method belongs to stochastic descent methods. Every individual solution from the final population is considered as an initial solution for the KA, which has an empirical diversification tool, but is a fast and simple local optimizer.

The KA is an approximation technique based on stochastic annealing (Fleury, 1995), inspired by the simulation annealing method, but having a quite different searching strategy. A detailed description of the KA is given in Minzu and Henrioud (1998). As in the case of simulated annealing, the “Kangaroo” method is implemented by an iterative procedure that minimizes an objective function $f(u)$. A current solution $u$ of the considered problem is replaced by a better one, situated in its neighborhood $N(u)$, using a random selection. The algorithm tries “$A$” times to improve the current solution, where $A$ is a parameter of the algorithm. If a new improvement is no longer possible, a “jump” procedure is performed, in order to escape from the attraction of a local minimum. This time the improvement of the current solution is not compulsory. This procedure can use a different neighborhood definition $N(u)$ for the random jumping. The best solution $u^*$ encountered in the iterative process is memorized. At the end of KA, $u^*$ is the “optimal” solution proposed by the algorithm. In this implementation, the stop criterion is a maximum iteration number. A parallel KA is implemented using a memory module for the storage of the best solutions found by every KA. The global best solution is determined by the evaluation of these solutions.

A test application of the general optimization algorithm implementing the parallel hybrid metaheuristic was realized for both the SMSP and TWA problem.

2. Single-machine scheduling problem

For a given processor and a set of jobs that must be executed on this processor, the problem is to determine the sequence of jobs such that the weighted tardiness is minimized. The tardiness of a job $j$ ($j = 1, \ldots, n$) is calculated using the attributes of the jobs: $p_j$ the processing time, $d_j$ the due date and $w_j$ the penalty liable for each unit of delay. If $u$ is a solution of the problem, that is a permutation of the job set, the total weighted tardiness is

$$f(u) = \sum_{j=1}^{n} w_j T_j.$$

The optimality criterion of the SMSP is

$$\min_u f(u).$$

For SMSP, the solutions are encoded by a natural representation (Michalewicz, 1992). In this case, each gene represents the job number, and the gene position in the chromosome represents the job position in the solution (sequence). In this way, in a chromosome, there can be found the sequence that defines the job’s scheduling. The chromosome dimension is equal to $n$, the number of jobs.

The start point for generating the initial population is an initial solution, which is established by the earliest due date rule. In this way, the initial solution contains all the jobs sorted after their correspondent due date. This is a good solution, but it does not take into account the influence of the penalties (a small delay may be penalized a lot). Then, the initial population is generated by a procedure called randomized earliest due date (Madureira et al., 2001).
In the case of SMSP, the implementation of the GA is done in a traditional manner for the scheduling problems (Minzu and Beldiman, 2003). At any iteration of the GA, four actions are performed as follows:

1. the $N$ solutions of the current population are subject to the crossover operator. Therefore, the offspring resulted, consisting in $K$ new solutions, is added to the population;
2. the mutation operator is applied to the $N+K$ solutions of the increased population;
3. the eventual duplicate solutions belonging to the increased population are subject to a new mutation. Thus, the number of duplicates is minimized.
4. the best $N$ solutions (from the point of view of the objective function) among the increased population are kept in the next population. So, the population’s dimension is maintained constant, equal to $N$, the size of the initial population.

A two-point crossover operator was used. Applied to a pair of solutions, the operator chooses two points in the first chromosome, and preserves its outer parts. The inner part is filled with the rest of the jobs, in the same order in which they appear in the second chromosome (Grefenstette, 1987). So, the validity of the offspring is guaranteed.

In order to avoid the attraction of a local optimum, that means the premature convergence of the genetic algorithm, an inversion mechanism was used as mutation operator. For example, the chromosome $[1\ 4\ 3\ 2\ 5]$ will become after mutation $[1\ 3\ 4\ 2\ 5]$, where the second and the fourth genes are subject of mutation.

While the chromosomes’ number $n$ represents the problem dimension, the value $N$ must be chosen so that it should be directly proportional to the solutions’ space dimension. In other words, the $N$ solutions belonging to the final population have to “cover” as uniformly as possible the solutions’ space. A possible value for tuning $N$ may be

$$N \approx 2n.$$  

(1)

Obviously, a higher value for $N$ may be adopted but the user must pay attention to the practical complexity of the GA. The other implementation parameter of the GA that must be chosen is $K$, the dimension of the offspring produced by the crossover operator and inserted in the current population. In this work, values of $K$ such as

$$\frac{N}{3} \leq K \leq \frac{N}{2}$$

were considered. A higher value of this parameter increases the computational time without improving the results.

Another implementation parameter is $N_{gen}$ the number of generations. A relation between $n$ and $N_{gen}$ must be found. The performed tests have proven that $N_{gen}$ might be considered directly proportional to $n$:

$$N_{gen} \approx an.$$  

The test has proved that the value $a = 2$ is sufficient, but $N_{gen}$ can take higher values in order to improve the spread of the solutions belonging to the final population in the solutions’ space.

The implementation of KA uses as neighborhood $N(u)$ the set of solution $u'$ obtained from $u$ by the permutation of the jobs placed on positions $i$ and $i+1$. For example, if $u = [1\ 4\ 3\ 2\ 5]$, it holds

$$N(u) = \{[4\ 1\ 3\ 2\ 5],[1\ 3\ 4\ 2\ 5],[1\ 4\ 2\ 3\ 5],[1\ 4\ 3\ 5\ 2],[5\ 4\ 3\ 2\ 1]\}.$$  

In the case of SMSP, a possible definition of the neighborhood $N(u)$ is the whole search space, but the KA slowly converges (with probability 1) to the global optimum. A very important aspect is the fact that deterministic heuristics may be integrated in the “jump” procedure, in order to guide the search of an optimum solution. The convergence of the KA is kept, if the accessibility constraint is met (Minzu and Henrioud, 1998). So the “jump” procedure determines the job with the biggest weighted tardiness and replaces it with a job situated on its left. In this way, there is a chance to diminish the value of the criterion $f(u)$ and to simultaneously leave the local optimum.

3. TWA assignment problem

The assignment problem consists in dispatching the tasks to the available workstations. A TWA must conform to the precedence constraints between tasks, involved by the process. At the same time, it must be feasible, i.e. each task must be liable to be performed by the assigned workstation.

The set $Q$ of $N$ tasks, the set of $M$ operational workstations and the digraph corresponding to the precedence relation defines the TWA problem. A proper assignment is a feasible partition $u$ that meets the precedence constraints:

$$u = \{P_1, P_2, \ldots, P_M\}, \quad \text{with } P_i \subset Q, i = 1, \ldots, M.$$  

A TWA assignment has to meet the precedence constraints, i.e.

$$(t_{j_1} \text{ precedes } t_{j_2}) \Rightarrow i_{j_1} \leq i_{j_2}, \quad \forall t_{j_1}, t_{j_2} \in Q, j_1 \neq j_2,$$

with $t_{j_1} \in P_{i_{j_1}}, t_{j_2} \in P_{i_{j_2}}$.  

(2)

Generally, there are many proper assignments. One may choose the assignment that minimizes an objective function. A possible optimality criterion is to minimize the cycle time of the manufacturing line (the line’s cycle time is the maximum work content of a workstation). So, TWA is an optimization problem that is obviously NP-hard because it is a graph-partitioning problem (Glover et al., 2004), which satisfies an optimality criterion.

A possible solution encoding, used in this work, is a vector $u$ with $n$ elements, such that

$$u(i) = j, \quad i = 1, \ldots, n \quad \text{and} \quad j \in \{1, \ldots, M\},$$

which means that the task $i$ is assigned to the workstation $j$. 

The main difference between the TWA problem and SMSP is the existence of the precedence constraints. This involves some difficulties concerning the implementation of genetic operators, such as crossover and mutation. The precedence constraints are specified through a precedence digraph as shown in Fig. 2. This one corresponds to a TWA problem with \( n = 16 \) tasks and \( M = 4 \) workstations.

Also a two-point crossover operator was used. For the problem described in Fig. 2, let two chromosomes subject of crossover operator be

\[
\begin{align*}
00010 & | 11111 | 122233, \\
00010 & | 11211 | 222233, 
\end{align*}
\]

where the four workstations assigned to the tasks were specified from 0 to 3 and the character “\(|\)“ marks a fracture point used by the crossover operator. These two chromosomes encode the following two assignments:

- \( P_0 = \{0, 1, 2, 4\}, P_1 = \{3, 5, 6, 7, 8, 9, 10\} \);
- \( P_2 = \{11, 12, 13\}, P_3 = \{14, 15\} \);
- \( P'_0 = \{0, 1, 2, 4\}, P'_1 = \{3, 5, 6, 8, 9\} \);
- \( P'_2 = \{7, 10, 11, 12, 13\}, P'_3 = \{14, 15\} \).

The offspring resulted after the crossover is

\[
\begin{align*}
00010 & | 11211 | 122233, \quad \text{(3)}
\end{align*}
\]

which encodes the assignment:

- \( R_0 = \{0, 1, 2, 4\}, R_1 = \{3, 5, 6, 8, 9, 10\} \);
- \( R_2 = \{7, 11, 12, 13\}, R_3 = \{14, 15\} \).

According to the notation made in the constraint (2), it holds:

- \( j_1 = 7, \ j_2 = 10, \ i_1 = 2, \ i_2 = 1 \);
- \( j_1 \) precedes \( j_2, \ i_1 > i_2 \).

It can be easily observed that this is not a proper assignment because the precedence constraints are violated.

Generally speaking, the nature of the chosen crossover operator makes possible the precedence constraint not to be met. Hence, the correction of the generated offspring becomes compulsory. The correction method suggested in this paper is described in the sequence. Every time the precedence constraint for the tasks \( j_1 \) and \( j_2 \) is violated, the task \( j_2 \) is assigned to the workstation of the task \( j_1 \). So, the correction rule is:

\[
(j_1 \text{ precedes } j_2) \quad \text{and} \quad (i_1 > i_2) \Rightarrow i_2 = i_1. \quad \text{(4)}
\]

Applying this rule to chromosome (3), the final result of the crossover operator becomes

\[
00010 | 11211 | 222233.
\]

The implementation of the proposed correction method supposes that all the tasks \( j_2 \) are visited and for all of its predecessors \( j_1 \) constraint (2) is verified. If the tasks \( j_2 \) will be reviewed in the order \( j_2 = 0, 1, \ldots, n-1 \), it is possible that the correction method fails as in the following example. In Fig. 2, a fragment of a precedence graph is shown together with the assignment of tasks 4–6. The correction method supposes the following steps:

- The task 4 is inspected and all its predecessors and the precedence constraint is supposed to be met. Thus its assigned workstation remains unchanged (2).
- The task 5 is reviewed and the tasks 6 and 5 met constraint (1). Consequently its assigned workstation remains unchanged (1).
- The task 6 is visited and, according to the correction rule (4), its assigned workstation is modified from 1 to 2. But this adjustment rises a problem: the precedence constraint between tasks 6 and 5 is not still met.

Fig. 2. An example of precedence graph.
This problem can be avoided if the tasks are inspected in such an order that any task is visited after all its predecessors are reviewed before. In this work, such an order is predetermined in the initialization phase of the GA taking into account the precedence graph of the treated problem. Using this order, the assignment for the example from Fig. 2 becomes \([\ldots 2, 2, 2, \ldots]\) instead of \([\ldots 2, 2, 1, \ldots]\), the sequence \(\ldots 4, 6, 5, \ldots\) being an adequate order.

In this paper, the proposed mutation operator alters a randomly chosen gene of the chromosome, by increasing or decreasing the value of this gene with one unit. Because the value of the gene means the workstation to which the task is assigned, the mutation performs a shifting of the task to a neighbor workstation situated to the left or to the right of the current workstation. The problem is that a mutation may perform an illegal movement from the point of view of the precedence constraint. In order to avoid the violation of this constraint, the shifting of a task must be allowed only after the meeting of the precedence constraint in the new chromosome. An outline of the proposed mutation operator is given hereafter:

```plaintext
repeat
  • random selection of a gene;
  • try to shift the task to the left (decreasing by 1 the value of the gene). If the precedence constraint is met, acknowledge the shifting and jump over the next step.
  • try to shift the task to the right (increasing by 1 the value of the gene). If the precedence constraint is met, acknowledge the shifting.
until a shifting is done
```

Obviously, there is always a task in the assignment that can be shifted, thus the loop terminates.

The implementation of the KA needs the definition of the neighborhood function \(N(u)\). In order to define it, two shift operators are considered in Minzu and Henrioud (1998). The shift-left and the shift-right operators move a given task to a neighbor workstation if the precedence constraints are met. The neighborhood \(N(u)\) is the set of solutions \(u^\prime\) obtained from \(u\) by applying the shift-left or the shift-right operator. A shift operator is well defined if it preserves the meeting of precedence constraint. The main idea is that the next solution is obtained from the current one by a well-defined shift operator. The convergence speed is improved by a deterministic heuristic that guides the “descent” procedure. Thus, \(N(u)\) is obtained considering only the move of a task belonging to the maximum work content workstation. So, the number of solutions belonging to the current neighborhood is drastically reduced. That is why the estimation \(A = 0.7\ \text{card}(N)\) recommended in Fleury (1995) is no longer necessary and practical tests have proved that a value \(A = 6-10\) is sufficient.

The neighborhood \(N(u)\) used by the jump procedure is considered to be equal to \(N(u)\). The move of only one task in a neighbor workstation may radically change the value of the optimality criterion (the cycle time) and may place the current solution in another region of the solutions' space. The practical tests have proved that this is sufficient for leaving the local minimum by a well-defined move.

In this work, the initial population generation was implemented in a very simple manner. It needs only an initial solution \(u_0\) given by the user. The other \(N-1\) solutions of the initial population are obtained through an iterative process that yields a new solution at each iteration, using the same procedure as the mutation operator described before. Obviously, the use of this procedure is just a possibility and any other method that gives \(N\) different solutions may be used.

### 4. Computational results

Computational tests were performed in order to compare the proposed parallel metaheuristic hybrid system with a stand-alone GA. The software developed was coded in C and the tests were performed on a PC Pentium IV at 1400 MHz. As stop criterion, an upper limit for the number of generations, \(N_{\text{gen}}\) (for the GA), and for the number of iterations (for the KA) was used.

For example, the two algorithms were applied to the same instances of SMSP. A set of problems with 70 jobs was considered (OR library). Table 1 presents the results obtained using only the stand-alone GA. For this test, it was chosen \(N_{\text{gen}} = 300\). In the second test, the hybrid system ran with \(N_{\text{gen}} = 50\) generations for the GA, and 200, 500 and 1000 iterations for the KA. The results are shown in Table 2. The column “Best GA value” contains the best value of the objective function at the end of the first phase (GA), while the column “Best KA value” gives the final best value (at the end of parallel KA). Because the optimal values were known for these problems, the deviation from the optimal value could be calculated. The optimal values were found for most of the problems. The best values of the optimality criterion are presented in bold characters.

Comparing the two tables, one can see that the hybrid system is more efficient. Despite the fact that the stand-alone GA evolves during 300 generations (in 42 s), it does not converge in the first phase (GA). The results obtained using the proposed hybrid system are obviously better. The table shows the best values obtained by the hybrid system at the end of parallel KA. The optimization of the evolutionary system is performed with the same instances of SMSP. A set of problems with 70 jobs was considered (OR library).

<table>
<thead>
<tr>
<th>Problem</th>
<th>Optimal solution</th>
<th>Best value</th>
<th>Deviation of the best value (%)</th>
</tr>
</thead>
<tbody>
<tr>
<td>wt70a</td>
<td>140</td>
<td>185</td>
<td>32</td>
</tr>
<tr>
<td>wt70b</td>
<td>616</td>
<td>853</td>
<td>38.4</td>
</tr>
<tr>
<td>wt70c</td>
<td>826</td>
<td>915</td>
<td>10.7</td>
</tr>
<tr>
<td>wt70d</td>
<td>1176</td>
<td>1368</td>
<td>16.3</td>
</tr>
<tr>
<td>wt70e</td>
<td>658</td>
<td>707</td>
<td>7.4</td>
</tr>
<tr>
<td>wt70f</td>
<td>336</td>
<td>468</td>
<td>39.2</td>
</tr>
<tr>
<td>wt70g</td>
<td>546</td>
<td>758</td>
<td>38.8</td>
</tr>
<tr>
<td>wt70h</td>
<td>546</td>
<td>711</td>
<td>30.2</td>
</tr>
</tbody>
</table>
not reach the same results as the hybrid system. The latest evolves during 50 generations (in 7 s) for the GA, before applying the KA for 2000 iterations (in 4 s). So, in a shorter period of time, about a quarter, the hybrid system finds practically the optimal solution.

Benchmark problems from OR library were solved with the hybrid system and the results are presented in Table 3. A good behavior of the hybrid system can be confirmed. The hybrid system was tested also with TWA problems, whose name is coded by twa\(x\)-\(y\), where \(x = n\) and \(y = M\). These instances were generated offline using a repetitive concatenation of structures (for example, the one presented in Fig. 2) whose optimal assignments can be calculated. Therefore, this regularity allows knowing the optimal value while keeping the difficulty of the problem. Table 4 presents the results of these tests for \(A = 6\) value, which was experimentally tuned.

Another test was performed in order to study the behavior of the hybrid system for different values of \(A\) (Table 5). In principle, a higher value for \(A\) improves the intensification character of the KA. As mentioned before, the “descent” procedure does not behave purely randomly and the size of the current neighborhood is reduced. Hence, the KA had already converged to the local minima in a small number of iterations (less than 1000). So, the intensification character is satisfactory with \(A = 6\).

The tuning of the number of solutions in the evolving population, \(N\), is quite difficult. The results presented in Table 6 are obtained by setting \(N\) to 50, 100 and 200, and letting the GA evolve during 200 generations. The optimization algorithm was applied to the 100 tasks with 20 workstation problems. The results show that the KA (500 iterations with \(A = 6\)) works very well with a population containing \(N = 200\) solutions, which is a confirmation for the empirical relation (1). The total execution time of the hybrid system is under 24 s for all the tested instances.
Table 5
Behavior of hybrid system for different values of $A$

<table>
<thead>
<tr>
<th>Problem</th>
<th>Best GA value after 1000 iter.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$A = 6$</td>
</tr>
<tr>
<td>twa60-15a</td>
<td>115</td>
</tr>
<tr>
<td>twa60-15b</td>
<td>135</td>
</tr>
<tr>
<td>twa60-15c</td>
<td>445</td>
</tr>
<tr>
<td>twa80-20a</td>
<td>290</td>
</tr>
<tr>
<td>twa80-20b</td>
<td>190</td>
</tr>
<tr>
<td>twa80-20c</td>
<td>290</td>
</tr>
<tr>
<td>twa100-20a</td>
<td>290</td>
</tr>
<tr>
<td>twa100-20b</td>
<td>400</td>
</tr>
<tr>
<td>twa100-20c</td>
<td>720</td>
</tr>
<tr>
<td>twa120-24a</td>
<td>290</td>
</tr>
<tr>
<td>twa120-24b</td>
<td>400</td>
</tr>
<tr>
<td>twa120-24c</td>
<td>720</td>
</tr>
</tbody>
</table>

Table 6
Behavior of the hybrid system for different sizes of the evolving population

<table>
<thead>
<tr>
<th>Problem</th>
<th>Best GA value for $N = 50$</th>
<th>Best KA value for $N = 50$</th>
<th>Best KA value for $N = 100$</th>
<th>Best KA value for $N = 200$</th>
</tr>
</thead>
<tbody>
<tr>
<td>twa100-20a</td>
<td>290</td>
<td>270</td>
<td>250</td>
<td>160</td>
</tr>
<tr>
<td>twa100-20b</td>
<td>400</td>
<td>400</td>
<td>400</td>
<td>320</td>
</tr>
<tr>
<td>twa100-20c</td>
<td>720</td>
<td>720</td>
<td>720</td>
<td>660</td>
</tr>
</tbody>
</table>

5. Conclusion

The paper has proposed a parallel metaheuristic hybrid system for solving discrete optimization problems. Two real problems from the manufacturing field were solved with the hybrid system. A GA performs the first phase, while a parallel stochastic descent method, called ‘Kangaroo’, performs the intensification phase. The KA is a very simple and efficient intensifier.

Two real problems in the manufacturing field were solved using the proposed metaheuristic. This was the opportunity to underline some aspects regarding the implementation of this hybrid system. The impact of the precedence constraints upon the implementation of the genetic operators (crossover and mutation) was also emphasized.

The computational results obtained during different tests performed with a software implementation have proved a satisfactory behavior and a very good practical complexity. The functioning of this system was compared to a stand-alone GA. The tests have proved that this structure is very efficient. The optimal solutions were found in a quite acceptable time and number of iterations. It was shown that the behavior of the proposed hybrid system is quite satisfactory.

The future work will be directed to the usage of some elements from scatter search or path relinking (Glover et al., 2004) instead of the GA, in order to have a population of solutions well spread in the solutions space after the first phase of the algorithm.

References

OR library (http://www.mscgna.ms.ic.ac.uk/jeb/orlib/wtinfo.html).