INTERNATIONAL JOURNAL OF COMPUTERS COMMUNICATIONS & CONTROL
ISSN 1841-9836, 9(6):741-748, December, 2014.

PARMODS: A Parallel Framework for MODS Metaheuristics

E.D. Nino Ruiz, S. Miranda, C.J. Ardila, W. Nieto

Elias D. Nino Ruiz*, Stella Miranda,
Carlos J. Ardila, Wilson Nieto
Universidad del Norte
Computer Science Department
Colombia, Barranquilla
{enino,stellam,carila,wnieto}@uninorte.edu.co
*Corresponding author: enino@uninorte.edu.co

Abstract: In this paper, we propose a novel framework for the parallel solution of combinatorial problems based on MODS theory (PARMODS) This framework makes use of metaheuristics based on the Deterministic Swapping (MODS) theory. These approaches represents the feasible solution space of any combinatorial problem through a Deterministic Finite Automata. Some of those methods are the Metaheuristic Of Deterministic Swapping (MODS), the Simulated Annealing Deterministic Swapping (SAMODS), the Simulated Annealing Genetic Swapping (SAGAMODS) and the Evolutionary Deterministic Swapping (EMODS) Those approaches have been utilized in different contexts such as data base optimization, operational research [1–3, 8] and multi-objective optimization. The main idea of this framework is to exploit parallel computation in order to obtain a general view of the feasible solution space of any combinatorial optimization problem. This is, all the MODS methods are used in a unique general optimization process. In parallel, each instance of MODS explores a different region of the solution space. This allows us to explore distant regions of the feasible solution which could not be explored making use of classical (sequential) MODS implementations. Some experiments are performed making use of well-known TSP instances. Partial results shows that PARMODS provides better solutions than sequential MODS based implementations.
Keywords: MODS, Combinatorial Optimization, Parallel Framework.

1 Introduction

Combinatorial Optimization (CO) is a branch of optimization in which problems can be represented (or reduced) to discrete structures. In this ramification, we find many problems related to operational research and networking fields. Moreover, since the number of possible solutions in this kind of problems increase exponentially with regard to the input parameters, their numerical solution can be very hard (or impossible) to obtain, for instance solving their mathematical formulations. Thus, two important considerations should be taken into in account when we want to solve CO problems: the number of solutions to consider is only a subset of the feasible solution space and the solutions should be obtained in a polynomial time. The first item addresses the necessity of having good solutions and the second one, demands the elapsed time to be small for the proposed implementation. However, those features are opposite, this means, when the number of solutions explored from the feasible solution space is small, the solution is obtained in short time but maybe, the approximated optimal solutions are not good enough. On the other hand, exploring more and more the feasible solution space provides better approximations to the optimal solution but, the performance of the method is affected considerably (long elapsed times). Thus, we need methods which in a polynomial time consider more and more solutions from the feasible solution space. Notice, we are not taking about exhaustive methods such as brute force but, combining information from different metaheuristics in a polynomial time which can be done in parallel.

This paper is organized as follows: in section 2 the TSP problem is introduced and MODS metaheuristics are presented, section 3 describes the proposed implementation and, section 4 and 5 provide the experimental results and conclusions, respectively.

2 Preliminaries

Following the previous section, one of the most widely used CO problems is the Traveling Salesman Probem (TSP) Its importance is derived owing to its application to different branch and fields from optimization. Moreover, some well-known problems such as the Vehicle Routing Problem and the Transportation Problem are derived from the TSP formulation. In general, this problem is defined as follows: we have a set of N cities $\mathcal{C} = \{c_1, c_2, \ldots, c_N\}$, a matrix of weights $\mathbf{W} \in \mathbb{R}^{N \times N}$ whose elements $w_{i,j}$ provides the weight of going from c_i to c_j, for $1 \leq i, j \leq N$ and, the cost function

$$\mathcal{J}(\boldsymbol{\alpha}) = w_{N,\alpha_1} + \sum_{i=1}^{N-1} w_{\alpha_i,\alpha_{i+1}}, \tag{1}$$

which is subjected to

- Visiting each city from $\mathcal{C}$ once.
- Coming back to the initial city once the path $\boldsymbol{\alpha}$ has been completed.

Rudely speaking, we want to find the optimal path $\boldsymbol{\alpha}^*$ which provides the optimal components $w_{\alpha_i^*,\alpha_{i+1}^*}$, for $1 \leq i \leq N-1$, from $\mathbf{W}$ such that (1) is minimized. Note that, the number of possible solutions for the TSP problem increases by $N!$ with regard to the number of cities N.

As we mentioned before, the numerical solution of CO problems can be an exhaustive work making use of numerical methods and the TSP problem is not the exception. Note that, the TSP problem can be seen as a linear programming problem therefore well-known numerical methods based on Integer Programming and Simplex Methods could be used in order to solve (1) but, they have been proved to fail when the number of cities is large, as is usual in practice. On the other hand, the optimal solution of the TSP problem can be approximated making use of metaheuristics, we address one set of them in this paper, those are based on the Metaheursitic Of Deterministic Swapping (MODS) MODS is a metaheuristic inspired on the Automata Theory. Its application ranges from the Operational Research field to the Database Query Optimization area [7]. It is very important to note that, MODS methods are not novel methods, in general, they are nothing but classical combinatorial optimization methods represented on Deterministic Finite Automata structures. This improves the manner to design the solution of the problem since the optimial solution space and the transition between solutions (states of the automata) are defined prior any optimization process. This avoids, for instance, to explore unfeasible regions of the solution space.

MODS considers the next Deterministic Finite Automata (DFA)

$$\mathcal{Q}_{MODS} = \{\mathcal{S}, \Sigma, \delta, \mathcal{S}_0, \mathcal{J}\}\ , \tag{2}$$

where $\mathcal{S}$ is the feasible solution space, Σ is the input alphabet which is utilized by $\delta : \mathcal{S} \rightarrow \mathcal{S}$ in order to perturb the solutions, $\mathcal{S}_0$ contains the initial solutions and, $\mathcal{J}$ is the cost function to be optimized. The $\mathcal{S}$ space is unknown since it contains all the possible solutions of the CO problem. Putting all in the TSP context, $\mathcal{S}$ contains all the possible paths, $\mathcal{S}_0$ provides the initial path, $\mathcal{J}$ is the cost function (1) and Σ and δ provides all the possible manners to perturb a given path, for instance, given the path $\boldsymbol{\alpha}' = [1, 2, 3, 4]$ and the duple $\sigma_1 = (2, 3) \in \Sigma$ then, $\delta(\boldsymbol{\alpha}', \sigma_1) = [1, 3, 2, 4]$. The MODS metaheuristic is defined in Algorithm 2.

Algorithm 2 MODS Metaheuristic

```
Require: $\Sigma, \delta, \mathcal{S}_0, \mathcal{J}$
Ensure: $\boldsymbol{\alpha}^+ \approx \boldsymbol{\alpha}^*$
 1: $\boldsymbol{\alpha}^+ \leftarrow s \in \mathcal{S}_0$
 2: for $k = 1 \rightarrow M$ do
 3:   $\sigma_k \leftarrow \sigma \in \Sigma$
 4:   $\boldsymbol{\alpha}^- \leftarrow \delta(\boldsymbol{\alpha}^+, \sigma_k)$
 5:   if $\mathcal{J}(\boldsymbol{\alpha}^-) < \mathcal{J}(\boldsymbol{\alpha}^+)$ then
 6:     $\boldsymbol{\alpha}^+ \leftarrow \boldsymbol{\alpha}^-$
 7:   end if
 8: end for
```

Algorithm 3 SAMODS Metaheuristic

```
Require: $\Sigma, \delta, \mathcal{S}_0, \mathcal{J}, T_0, \rho, L$
Ensure: $\boldsymbol{\alpha}^+ \approx \boldsymbol{\alpha}^*$
 1: $\boldsymbol{\alpha}^+ \leftarrow s \in \mathcal{S}_0$
 2: for $k = 1 \rightarrow M$ do
 3:   for $i = 1 \rightarrow L$ do
 4:     $\sigma_i \leftarrow \sigma \in \Sigma$
 5:     $\boldsymbol{\alpha}^- \leftarrow \delta(\boldsymbol{\alpha}^+, \sigma_i)$
 6:     if $\mathcal{J}(\boldsymbol{\alpha}^-) < \mathcal{J}(\boldsymbol{\alpha}^+)$ then
 7:       $\boldsymbol{\alpha}^+ \leftarrow \boldsymbol{\alpha}^-$
 8:     else
 9:       Generate (uniformly) $\eta \in [0, 1]$
10:       Compute
              $\gamma = \exp\left(-\frac{\mathcal{J}(\boldsymbol{\alpha}^+) - \mathcal{J}(\boldsymbol{\alpha}^-)}{T_k}\right)$   (3)
11:       if $\eta < \gamma$ then
12:         $\boldsymbol{\alpha}^+ \leftarrow \boldsymbol{\alpha}^-$
13:       end if
14:     end if
15:   end for
16:   $T_{k+1} \leftarrow \rho \cdot T_k$
17: end for
```

SAMODS is a Simulated-Annealing (SA) based MODS method which explores the feasible solution space $\mathcal{S}$ in a more "generous" manner. It allows bad solution to be accepted in small optimization intervals (usually at the beginning of the iterations) Alike MODS, SAMODS makes use of the DFA

$$\mathcal{Q}_{SAMODS} = \{\mathcal{S}, \Sigma, \delta, \mathcal{S}_0, \mathcal{J}, T_0, \rho, L\} , \tag{4}$$

where $\mathcal{S}$, Σ, δ, $\mathcal{S}_0$ and, $\mathcal{J}$ remain unchanged, T_0 is the initial temperature, ρ is the cooling factor and, L is the number of refinement iterations. Note that, MODS accepts a new solution only when its optimal value is better than the current value (from the current path) On the other hand, SAMODS makes use of the Boltzamm distribution (5) in order to give the chance of a bad solution to be improved. This may provides a better solution than the best solution considered so far. Thus, at the beginning of the iterations, the number of solutions accepted as good is large but, this number is decreased when the iterations draws on since the parameter T_k is large and then, the condition of line 11 in Algorithm 3 is almost never satisfied. Following the SA principles, SAGAMODS [5] is defined on the SAMODS method but, when a bad solution is rejected (line 11 in Algorithm 3), the solution is improved making use of Genetic Algorithms. The supporting automata of SAGAMODS method is defined as follows:

$$\mathcal{Q}_{SAGAMODS} = \{\mathcal{S}, \mathcal{S}_0, \mathcal{C}(s, r, k), F(s)\}$$

$\mathcal{S}$ and $\mathcal{S}_0$ remain unchanged from the previous methods. In addition, $\mathcal{C}(s_1, s_2, k)$ is the crossover operator where $s_1 \in \mathcal{S}$ and $s_2 \in \mathcal{S}$ are parents solutions. Likewise, k provides the cross point. SAGAMODS method is presented in the Algorithm 4.

Algorithm 4 SAGAMODS Metaheuristic

```
Require: $\Sigma, \delta, \mathcal{S}_0, \mathcal{J}, T_0, \rho, L$
Ensure: $\boldsymbol{\alpha}^+ \approx \boldsymbol{\alpha}^*$
 1: $\boldsymbol{\alpha}^+ \leftarrow s \in \mathcal{S}_0$
 2: for $k = 1 \to M$ do
 3:   for $i = 1 \to L$ do
 4:     $\sigma_i \leftarrow \sigma \in \Sigma$
 5:     $\boldsymbol{\alpha}^- \leftarrow \delta(\boldsymbol{\alpha}^+, \sigma_i)$
 6:     if $\mathcal{J}(\boldsymbol{\alpha}^-) < \mathcal{J}(\boldsymbol{\alpha}^+)$ then
 7:       $\boldsymbol{\alpha}^+ \leftarrow \boldsymbol{\alpha}^-$
 8:     else
 9:       Generate (uniformly distributed) $\eta \in [0, 1]$
10:       Compute
              $$\gamma = \exp\left(-\frac{\mathcal{J}(\boldsymbol{\alpha}^+) - \mathcal{J}(\boldsymbol{\alpha}^-)}{T_k}\right) \quad (5)$$
11:       if $\eta < \gamma$ then
12:         $\boldsymbol{\alpha}^+ \leftarrow \boldsymbol{\alpha}^-$
13:       else
14:         Generate integer number (uniformly distributed) $\beta \in [1, modelsize]$
15:         call $\mathcal{C}(\boldsymbol{\alpha}^+, \boldsymbol{\alpha}^-, \beta)$
16:       end if
17:     end if
18:   end for
19:   $T_{k+1} \leftarrow \rho \cdot T_k$
20: end for
```

EMODS [6] is an evolutionary MODS method which improves the solutions making use of evolutionary techniques (crossover and mutation). A complete taxonomy of SAGAMODS and EMODS methods can be read in [4, Chapter 4].

Now we are ready to present our parallel approach of MODS based methods.

3 Proposed Implementation

To start, consider an array of processors available at time t:

$$\mathbf{P} = [p_1, p_2, \dots, p_n], \tag{6}$$

where n is the number of processors. For simplicity, we avoid the use of time indexes. Moreover, we consider that the metaheuristics MODS, SAMODS, SAGAMODS and EMODS can be run independently at different processors. Thus, we want to split the number of available processors per the number of metaheuristics, this is:

$$jobs_{proc} = \frac{n}{4}. \tag{7}$$

Consider the initial solution $s \in S_0$, then we denote the next DFAs based on the index $1 \le i \le n$:

$$\mathcal{Q}_i = \begin{cases} \mathcal{Q}_1 = \mathcal{Q}_{MODS} = \{\dots, s\} & \text{for } i = 1, 5, \dots \\ \mathcal{Q}_2 = \mathcal{Q}_{SAMODS} = \{\dots, s\} & \text{for } i = 2, 6, \dots \\ \mathcal{Q}_3 = \mathcal{Q}_{SAGAMODS} = \{\dots, s\} & \text{for } i = 3, 7, \dots \\ \mathcal{Q}_4 = \mathcal{Q}_{EMODS} = \{\dots, s\} & \text{for } i = 4, 8, \dots \end{cases}, \tag{8}$$

and then, we are ready to launch different processes based on the next rule

$$job_i = \begin{cases} \mathcal{P}(\mathcal{Q}_1, s) & \text{for } i = 1, 5, \dots \\ \mathcal{P}(\mathcal{Q}_2, s) & \text{for } i = 2, 6, \dots \\ \mathcal{P}(\mathcal{Q}_3, s) & \text{for } i = 3, 7, \dots \\ \mathcal{P}(\mathcal{Q}_4, s) & \text{for } i = 4, 8, \dots \end{cases}, \tag{9}$$

where the i-th process job_i ($\mathcal{P}(.,.)$) is executed in the processor p_i of (6), for $1 \le i \le n$. Note that, in (8) we choose the automata to be utilized and in (9), we launch the process. For instance, MODS metaheuristic is executed on processors 1,4,..., likewise, SAMODS is executed on processors 2,5,... and so on.

Denote by s_1, s_2, s_3, and s_4 the approximated optimal solutions provided by MODS, SAMODS, SAGAMODS and EMODS among processors, respectively, this is

$$\begin{aligned} s_1 &= \arg\min_{s^{(i)}_{MODS}} \left\{ \mathcal{J}\left(s^{(i)}_{MODS}\right), \text{ for } i = 1, 4, \dots \right\} \\ s_2 &= \arg\min_{s^{(i)}_{SAMODS}} \left\{ \mathcal{J}\left(s^{(i)}_{MODS}\right), \text{ for } i = 2, 5, \dots \right\} \\ s_3 &= \arg\min_{s^{(i)}_{SAGAMODS}} \left\{ \mathcal{J}\left(s^{(i)}_{MODS}\right), \text{ for } i = 3, 6, \dots \right\} \\ s_4 &= \arg\min_{s^{(i)}_{EMODS}} \left\{ \mathcal{J}\left(s^{(i)}_{MODS}\right), \text{ for } i = 4, 7, \dots \right\} \end{aligned}$$

where, for instance, $s_1^{(1)}$ is the approximated optimal solution of MODS from the processor 1. Then, we choose the best approximation,

$$s^+ = \arg\min_{s_i} \{\mathcal{J}(s_k), \text{ for } 1 \le k \le 4\} \tag{10}$$

which will serve as the new initial solution in $\mathcal{S}_0$. This iterative process is called PARMODS (Parallel MODS) and it is summarized in Algorithm 5. Note that, the required components of this metaheuristic varies from the definitions of the automatas, that is why the common components are shown in the inputs and the optional parameters are expressed by dots.

Algorithm 5 PARMODS Metaheuristic

```
Require: $\Sigma, \delta, \mathcal{S}_0, \mathcal{J}, \ldots$
Ensure: $\boldsymbol{\alpha}^+ \approx \boldsymbol{\alpha}^*$
 1: $\boldsymbol{\alpha}^+ \leftarrow \mathcal{S}_0$
 2: for $t = 1 \to M$ do
 3:   for all $i = 1 \to n$ do
 4:     Launch $job_i$ according to (9).
 5:   end for
 6:
        $$\boldsymbol{\alpha}^- = \arg\min_{s_{*MODS}^{(j)}} \left\{ \mathcal{J}\left(s_{*MODS}^{(j)}\right)\ ,\ \text{for } 1 \le j \le n \right\}$$
 7:   if $\mathcal{J}(\boldsymbol{\alpha}^-) < \mathcal{J}(\boldsymbol{\alpha}^+)$ then
 8:     $\boldsymbol{\alpha}^+ \leftarrow \boldsymbol{\alpha}^-$
 9:   end if
10: end for
```

Notice, the computational cost of the method per iteration will be given by the number of iterations of PARMODS times the upper bound

$$\mathcal{O}\left(A_{PARMODS}\right) = \max\left(\mathcal{O}(A_{MODS}), \mathcal{O}(A_{SAMODS}), \mathcal{O}(A_{SAGAMODS}), \mathcal{O}(A_{EMODS})\right)\ ,$$

where the letter A counts for “Algorithm". Note that, since all the methods are executed in parallel, the computational effort of PARMODS is provided by the largest upper bound, which, in general, is provided by SAGAMODS.

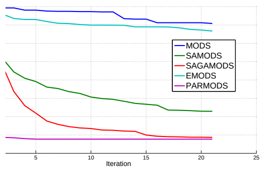
4 Experimental Results

We study the performance and efficiency of PARMODS making use of TSP instances from the TSPLIB. The selected TSP instances are KROA100 and KROA150 which contain 100 and 150 cities, respectively. The solutions obtained by the methods are presented in Table 1 and figure 1.

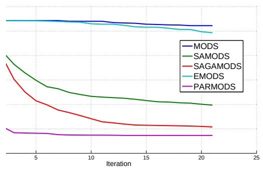
Metaheuristic	Processors	$\mathcal{J}$ KROA100	$\mathcal{J}$ KROA150
MODS	N/A	1.6165	2.6122
SAMODS	N/A	0.6583	0.9865
SAGAMODS	N/A	0.3739	0.5399
EMODS	N/A	1.5340	2.4663
PARMODS	4	0.3545	0.3664
PARMODS	8	0.2827	0.3359
PARMODS	12	0.2827	0.3359

Table 1: Cost function values $\times 10^5$ for different MODS implementations.

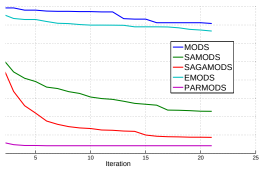
In figure 1 can be seen how PARMODS outperforms the other MODS implementations (sequential MODS, SAMODS, SAGAMODS and EMODS) in terms of accuracy. Moreover, the MODS implementations are divided evenly onto the number of processors available (n) For instance, four processors means one instance of MODS, SAMODS, SAGAMODS and EMODS are used at each processor when PARMODS is executed. Notice, PARMODS do not make use of parallel resources in order to split the domain but to obtain information about the feasible solution space. Since PARMODS spread MODS instances among processors, the best solution is used in the next generation of each MODS implementation (initial state of each Automata).



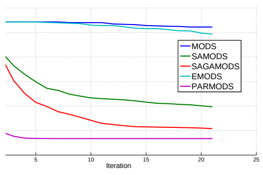
(a) KROA100 $n = 4$



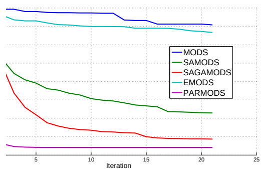
(b) KROA150 $n = 4$



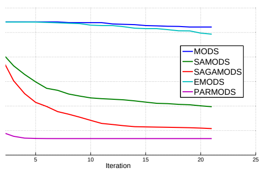
(c) KROA100 $n = 8$



(d) KROA150 $n = 8$



(e) KROA100 $n = 12$



(f) KROA150 $n = 12$

Figure 1: Graphical comparison of the cost function values per iteration for the different MODS implementations.

5 Conclusions

We propose a novel parallel method based on MODS theor5522894y. The proposed implementation exploits the attractive features of each MODS implementation. Initial results show that PARMODS provides better results among the compared methods. Moreover, when the number of processors is increased, the results are improved. However, we note that the results obtained for 8 and 12 processors are the same. This motivates to study theoretical bounds regarding the number of processors and the percentage of improvement on the solutions.

Bibliography

[1] Anonnimus (1964); Operational research studies in inventory sequencing simulation, *Production Engineer*, 43(9):437–438, DOI: 10.1049/tpe.1964.0060.

[2] Anonnimus (1964); Operational research studies. project a-inventory. *Production Engineer*, 43(9):438–447, DOI: 10.1049/tpe:19640061.

[3] Junyi Chen and Pingyuan Xi (2010); Simulation and application on modern operational research. In *Computer and Automation Engineering (ICCAE), 2010 The 2nd International Conference on*, 4: 118–121.

[4] Elias D. Niño (2012); *Real-World Applications of Genetic Algorithms*, chapter Evolutionary Algorithms Based on the Automata Theory for the Multi-Objective Optimization of Combinatorial Problems. InTech, Oxford, 2012. Book edited by Olympia Roeva.

[5] Elias D. Nino, Carlos J. Ardila, and Anangelica Chinchilla (2012); A novel, evolutionary, simulated annealing inspired algorithm for the multi-objective optimization of combinatorial problems. *Procedia Computer Science*, 9(0):1992 – 1998.

[6] Elias D. Nino-Ruiz (2012); Evolutionary Algorithm based on the Automata Theory for the Multi-objective Optimization of Combinatorial Problems. *International Journal Of Computers Communication & Control*, 7(5):916–923.

[7] Miguel Rodríguez, Daladier Jabba, Elias D. Niño, Carlos J. Ardila, and Yi-Cheng Tu (2013); Automata theory based approach to the join ordering problem in relational database systems. In Markus Helfert, Chiara Francalanci, and Joaquim Filipe, editors, *DATA*, pages 257–265. SciTePress.

[8] Li Zhengfeng and Ye Jinfu (2010); Study on the evolutionary mechanism from operational research activities to sustainable competitive advantage. In *Intelligent Computation Technology and Automation (ICICTA), 2010 International Conference on*, 3: 580–584.