A Solution to Multidimensional Knapsack Problem Using a Parallel Genetic Algorithm

Balderas_Posada Carlos, Vilariño_Ayala Darnes
Somodevilla_Garcia Josefa, León_Silverio Saúl, and Tovar_Vidal Mireya
charlie_kanon@hotmail.com, darnes@cs.buap.mx
mariasg@cs.buap.mx, saul.ls@live.com, mtovar@cs.buap.mx
doi:10.4156/ijiip.vol1. issue2.5

Abstract
The genetic algorithms are metaheuristic search algorithms based on natural selection mechanisms. The multidimensional knapsack problem (MKP), is a combinatorial optimization NP-Hard [1]. Several metaheuristics have been developed in order to find, if not the optimal solution, a "good solution", for this type of problem, although sometimes the endpoint is under local optimal premises. This paper proposes a parallel genetic algorithm (PGA) using GALib library to resolve, in particular, MKP medium-sized problems. The (PGA) enabled to resolve most of the problem tests and four of them approached the best reported value. A simplification algorithm also was developed, which allowed either to reduce the original problem size or to detect a priori a feasible solution for medium size problems. Besides, from the experiments can be concluded that the PGA showed the best performance using four nodes in the cluster.

Keywords: Parallel System, Genetic Algorithm, Multidimensional Knapsack Problem

1. Introduction

A good reason for adopting genetic algorithms (GA) in solving optimization problems is that they often require an adaptive program, able to behave well to changes in the environment. GA aims to design high quality solutions to highly complex problems and also to develop their ability to adapt these solutions to environmental changes [2].

The evolution can be seen also as a way of designing solutions to complex problems, with the ability to innovate. These are the reasons for which evolutionary mechanisms are a source of inspiration for searching algorithms. Proper functioning of a biological organism depends on many criteria, which also vary as the organism evolves. The mode of evolution is continuously “looking for” between a changing set of possibilities. Therefore, it can be considered a massively parallel search method, measuring and change millions of species in parallel [3].

Within the mathematical programming appear very often, problems in which the variables involved take only 0-1 values, i.e. binary variables. The general formulation is as shown in Problem (1).

\[
\min (\max) \quad z = c^T x \\
\text{s.t.} \quad Ax \leq b
\]

If the coefficients in the matrix \( A \) are all positive, it is called the multidimensional knapsack problem (MKP), deeply touched by the research community, since the exact algorithms to solve MKP belong to the class NP-hard.

Exact algorithms have been proposed, the stronger is the Balas [4, 5, 6], who proposed an additive algorithm for solving such problems. Chu and Beasley [7] developed a very efficient metaheuristic, which first reported the concept of reparation for a solution, in order to quickly reach workable solutions. In [7] was also showed that the heuristic is able to get solutions very close to the optimal for problems of different characteristics that require a “small” amount of computational effort. However, the issues addressed so far have been considered medium in size, i.e. less than 1000 variables. ORLIB library [8] reported very good solutions for various test problems.
This work shows the development of a Parallel Genetic Algorithm implemented in OOMPI, which uses the GALib library to solve MKP problems [9, 10], with more than 1000 variables.

The article structure is as follows, in the introduction the nature of the problem to be solved is specified, and Genetic Algorithms section explains this technique and related terms. The implementation section addresses the Sequential Genetic Algorithm (SFA); the simplification algorithm is proposed and the Parallel Genetic Algorithm (PGA) developed is described with its proposed migration scheme. In the section on Testing and Analysis of results, the algorithms behavior for the test problems reported in the literature is exposed, and finally in the conclusions an analysis of the results is shown.

2. Genetic Algorithms

The first mention of the term "genetic algorithm" and its first publishing applications are due to Bagley in 1967 [11]. Bagley designed genetic algorithms to find sets of parameters in functions evaluation of games, and compared them with correlation algorithms which are modeled learning procedures. However, John Holland [12] is considered the creator of genetic algorithms for the years 1960 and 1970.

The genetic algorithm of Holland is a method to move from a population of chromosomes to a new population, using a similar system to natural selection with crossing, mutation and investment operators, inspired by genetics. In this primitive algorithm, each chromosome consists of genes (bits), and each of them is a sample of a particular allele (0 or 1).

Genetic algorithms, as already suggested, are Metaheuristic search algorithms based on mechanisms of natural selection and genetic, combining survival of the most compatible elements between the string structures (alleles), with information structure (gene) already created at random. These are combined to develop a search algorithm with some of the innovative capabilities of the human quest [13].

A Genetic Algorithm, as a simulation of the evolutionary theory, requires that biological mechanisms been move to the computational mechanisms. These mechanisms are:

- Population
- Inheritance
- Mutation
- Selection
- Adaptation
- Reproduction

To do so, a genetic algorithm should establish a set of components, which allow the evolution of a population with possible solutions for a given problem [2].

3. System Implementation

For the system implementation the programming environment C++ was used, this language was chosen for its power, being object-oriented and since libraries GALib and OOMPI are written under this language. This last feature is essential, for the facility that provides the programming paradigm to write, extend, maintain and reuse code.

For the design of algorithms concepts from GALib [9, 10] were used, which is a library of functions in C++ that provides the programmer with a set of objects for the development of genetic algorithms. With GALib is possible to solve optimization problems by constructing a genetic algorithm, which uses data structures and standard or specific selection, cross and mutation operators. The programmer creates a fitness function, which allows the genome to adapt to the results you want to find, in our case the objective function in Problem (1).
To implement the algorithm Parallel Message Passing Interface (MPI) was chosen. Object Oriented MPI (OOMPI) is a class library specification that encapsulates the functionality of MPI into a functional class hierarchy, providing a simple, flexible and intuitive interface.

3.1. Simplification Algorithm

Supported by the criteria proposed by Crowder, Johnson and Padberg [14], the following simplification algorithm is defined, which proposes a comprehensive treatment of system constraints, by applying different criteria, in order to detect without large computational effort, if the problem is or is not feasible solution. If it is possible obtain a feasible solution, the algorithm generates an outline of the solution, i.e. one's place in alleles of the chromosome bearing the algorithm to a feasible solution and 0's otherwise (see Algorithm 1).

Each constraint in Problem (1) can be separated according equation (2)

$$\sum_{j \in N^+} a_j x_j + \sum_{j \in N^-} a_j \leq a_0$$

where:
N+ denotes the set of indices with variables that have positive coefficient.
N- denotes the index set of variables that have a negative coefficient in each constraint.

Algorithm 1: Simplification Algorithm

Step 1: Initialize solution
\( f_j = \text{false} \)

Step 2: Feasibility Analysis
If for some constraint is satisfied that \( \sum a_j > a_0, j \in N^- \) then go to step 5.
If not, then go to Step 3

Step 3: Inactivity Analysis
If for some constraint is satisfied that \( \sum a_j \leq a_0, j \in N^- \) then remove the restriction.
If \( f_j = \text{false} \), then go to Step 4
If not, then go to Step 6

Step 4: Set Solution:
// In zero:
If true that for \( a_j > a_0 - \sum a_k \) then \( j \in N^+, k \in N^- \)
\( x_j = 0 \)
\( f_j = \text{true} \)
// In one:
If true that for \( -a_j > a_0 - \sum a_i \) then \( j \in N^-, k \in N^- \)
\( x_j = 1 \)
\( f_j = \text{true} \)
If \( f_j = \text{true} \) then go to Step 2
If not, then go to Step 6

Step 5: The problem has no feasible solution. Go to Step 6
Step 6: End of algorithm

3.2. Sequential Genetic Algorithm

The following describes the sequential genetic algorithm (SGA) that was developed using the GALib library (see Algorithm 2).

Algorithm 2: Sequential Genetic Algorithm

Step 1: Create the object containing the MKP problem to be solved
Step 1.1: Read the file containing the problem
Step 1.2: Simplify the problem (Simplified Algorithm)
  Step 1.2.1: Check the feasibility of the problem
  Step 1.2.2: Check the idle restrictions
  Step 1.2.3: Set to zeros and ones the solution scheme
Step 2: Create the object that carries the information of the genome
Step 3: Create the object that contains the genetic algorithm
  Step 3.1: Indicate how many populations there
  Step 3.2: Enter the number of individuals per population
  Step 3.3: Establish how many generations are considered in each evolution
  Step 3.4: Set the rate of mutations in evolution
Step 4: Initialize population randomly
Step 5: Evolve
  Step 5.1: Enter the Fitness function
  Step 5.2: Perform cross current genome (which carries the genetic algorithm)
    using the solution model obtained in Step 1.
  Step 5.3: Check if the genome resulting from Step 5.2 is a feasible solution
    If so, enter the genome to the list of feasible solutions
  Step 5.4: While not complete the process of evolution, return to Step 5.1
Step 6: Show the solution of the problem

3.3. Parallel Genetic Algorithm

The Parallel Genetic Algorithm (PGA) developed in this research is built on the established Sequential Genetic Algorithm. The communication scheme used in parallel algorithm is a master-slave and migration of population from one node to another is done using the neighbor nodes. The algorithm is as follows:

Algorithm 3: Parallel Genetic Algorithm

Step 1: Read the file containing the MKP problem.
Step 2: Apply Simplifying Algorithm to the problem.
Step 3: Set the necessary parameters for the application of genetic algorithm:
  population size, number of generations, mutation rate and
type of crosses.
Step 4: Generate the initial population using the object that contains the genetic algorithm of the library.
Step 5: While the genetic algorithm will not be completed:
  It combines the current genome evolution and the solution scheme
    (obtained by the simplification algorithm)
  Every 50 iterations (migrate):
    Each slave node i sends its genome slave to the node i +1.
    Each slave node i receives the i-1 genome, following receipt of the genome,
    mix it up with its population.
Step 6: Send the best solution to node 1.
Step 7: The master node receives the solutions from the slave nodes.
Step 8: Show the result of the MKP problem, including the solution vector and objective function value.
Figure 1 shows the parallel model used in the PGA, as you can see, it is working under a scheme of slave communication, and in turn, slaves communicate linearly. The work of each node was explained previously in the algorithm 3.

4. Results

Problems test [8] were used to validate the results obtained by the implemented algorithms. For the implementation of these the following parameters were taken:

- $n$ is the size of the problem
- The number of individuals per population is:
  - If $n \leq 100$ then the population is 250 elements,
  - Otherwise the population is $n \times 4$ elements
- The number of generations is:
  - If $n \leq 100$ then 180 generations,
  - Otherwise $(n \times 3) / 2$ generations
- The number of nodes in the cluster is 2, 4 and 6

Table 1 shows a comparison of the results reported in [8] with the results obtained by the SGA and the PGA, as well as time spent in finding a solution.

<table>
<thead>
<tr>
<th>MKP Problem</th>
<th>Solution Reported</th>
<th>Solution SGA</th>
<th>Time</th>
<th>Solution PGA</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>mkp_01</td>
<td>3800</td>
<td>3800</td>
<td>0.962</td>
<td>3800</td>
<td>4.211</td>
</tr>
<tr>
<td>mkp_03</td>
<td>4015</td>
<td>4015</td>
<td>1.593</td>
<td>4015</td>
<td>36.02</td>
</tr>
<tr>
<td>mkp_04</td>
<td>6120</td>
<td>6120</td>
<td>2.014</td>
<td>6120</td>
<td>52.31</td>
</tr>
<tr>
<td>mkp_05</td>
<td>12400</td>
<td>12400</td>
<td>2.544</td>
<td>12400</td>
<td>49.85</td>
</tr>
<tr>
<td>mkp_06</td>
<td>10618</td>
<td>10618</td>
<td>4.937</td>
<td>10618</td>
<td>106.8</td>
</tr>
<tr>
<td>mkp_07</td>
<td>16537</td>
<td>-</td>
<td>-</td>
<td>16483</td>
<td>109.97</td>
</tr>
<tr>
<td>mk_gk01b</td>
<td>3674</td>
<td>-</td>
<td>-</td>
<td>3674</td>
<td>320.48</td>
</tr>
<tr>
<td>mk_gk02b</td>
<td>3869</td>
<td>-</td>
<td>-</td>
<td>3869</td>
<td>425.41</td>
</tr>
<tr>
<td>mk_gk03b</td>
<td>5506</td>
<td>-</td>
<td>-</td>
<td>5506</td>
<td>510.74</td>
</tr>
<tr>
<td>mk_gk04b</td>
<td>5610</td>
<td>-</td>
<td>-</td>
<td>5610</td>
<td>586.37</td>
</tr>
<tr>
<td>mk_gk05b</td>
<td>7344</td>
<td>-</td>
<td>-</td>
<td>7344</td>
<td>622.4</td>
</tr>
<tr>
<td>mk_gk06b</td>
<td>7481</td>
<td>-</td>
<td>-</td>
<td>7481</td>
<td>685.51</td>
</tr>
<tr>
<td>mk_gk07b</td>
<td>18578</td>
<td>-</td>
<td>-</td>
<td>18578</td>
<td>893.32</td>
</tr>
<tr>
<td>mk_gk08b</td>
<td>18285</td>
<td>-</td>
<td>-</td>
<td>18285</td>
<td>902.36</td>
</tr>
<tr>
<td>mk_gk09b</td>
<td>58085</td>
<td>-</td>
<td>-</td>
<td>56036</td>
<td>1858.23</td>
</tr>
<tr>
<td>mk_gk10b</td>
<td>57292</td>
<td>-</td>
<td>-</td>
<td>55741</td>
<td>2978.36</td>
</tr>
<tr>
<td>mk_gk11b</td>
<td>95231</td>
<td>-</td>
<td>-</td>
<td>93199</td>
<td>7896.24</td>
</tr>
</tbody>
</table>
A Solution to Multidimensional Knapsack Problem using a Parallel Genetic Algorithm
Balderas_Posada Carlos, Vilariño_Ayala Darnes

Figure 2. ORLIB [8] versus SGA and PGA

Figure 2 shows the values reported in ORLIB [8] and obtained by the algorithms developed in this paper. As you can see, from mkp_07, with more than 50 variables, SGA finds no solution. However, PGA does find a solution that quite resembles the best value reported so far.

Figure 3. MKP ORLIB with over 100 variables versus SGA and PGA

As shown in Figure 3, for problems with over 100 variables in library ORLIB, the SGA did not find any solution. As you can see the values reported in the first 8 problems were found by the PGA. However, in the last 3 problems, the solution obtained by the PGA is quite approaching best value reported so far.

Table 2 shows the time that took it for the PGA solution to the problem MKP, using 2, 4 and 6 nodes in the cluster.

Table 2. Time in seconds by the PGA with 2, 4 and 6 nodes in the cluster

<table>
<thead>
<tr>
<th>MKP Problem</th>
<th>Time in Sec. (2 Nodes)</th>
<th>Time in Sec. (4 Nodes)</th>
<th>Time in Sec. (6 Nodes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>mkp_01(*)</td>
<td>4.21</td>
<td>5.55</td>
<td>5.55</td>
</tr>
<tr>
<td>mkp_03</td>
<td>41.12</td>
<td>36.02</td>
<td>42.12</td>
</tr>
<tr>
<td>mkp_04</td>
<td>51.58</td>
<td>51.58</td>
<td>52.31</td>
</tr>
<tr>
<td>mkp_05</td>
<td>57.96</td>
<td>49.85</td>
<td>51.96</td>
</tr>
<tr>
<td>mkp_06</td>
<td>106.8</td>
<td>109.48</td>
<td>103.48</td>
</tr>
<tr>
<td>mkp_07</td>
<td>104.85</td>
<td>109.97</td>
<td>103.85</td>
</tr>
<tr>
<td>mk_gk01b(*)</td>
<td>302.92</td>
<td>327.92</td>
<td>320.48</td>
</tr>
<tr>
<td>mk_gk02b</td>
<td>448.05</td>
<td>425.41</td>
<td>403.05</td>
</tr>
<tr>
<td>mk_gk03b</td>
<td>510.74</td>
<td>515.49</td>
<td>517.49</td>
</tr>
<tr>
<td>mk_gk04b</td>
<td>595.73</td>
<td>586.37</td>
<td>618.73</td>
</tr>
<tr>
<td>mk_gk05b</td>
<td>607.51</td>
<td>632.51</td>
<td>622.4</td>
</tr>
<tr>
<td>mk_gk06b</td>
<td>610.21</td>
<td>685.51</td>
<td>664.21</td>
</tr>
<tr>
<td>mk_gk07b</td>
<td>893.32</td>
<td>890.6</td>
<td>808.6</td>
</tr>
<tr>
<td>mk_gk08b</td>
<td>904.14</td>
<td>902.36</td>
<td>902.14</td>
</tr>
<tr>
<td>mk_gk09b</td>
<td>1564.56</td>
<td>2438.56</td>
<td>1858.23</td>
</tr>
<tr>
<td>mk_gk10b</td>
<td>3487.31</td>
<td>2978.36</td>
<td>2243.31</td>
</tr>
<tr>
<td>mk_gk11b(*)</td>
<td>7896.24</td>
<td>7919.86</td>
<td>8726.86</td>
</tr>
</tbody>
</table>
(*) These problems are those set out in the graphic to be displayed below.

![Problems execution times using nodes 2, 4 and 6 nodes](image)

**Figure 4.** Problems execution times using nodes 2, 4 and 6 nodes

Figure 4 shows a run-time comparison of the results in seconds for the PGA with 2, 4 and 6 nodes; only for the mkp_01, which is the smallest analyzed problem (6 variables), the mk_gk01b has an average number of variables (100 variables) and the mk_gk11b with greater number of variables (2500 variables). Mkp_01 and mk_gk01b problems got the optimal solution, for the problem of over 1,000 variables, however only reported close to the optimal solution.

From the results, as you can see, the PGA reaches the better value reported in [8] using only four nodes in the cluster. This happens because 2 nodes seek best value obtained from two initial populations. However, with four nodes enriches space search to perform the migration and mix the best individuals of a population, so more quickly solve the problem lies. Besides, using 6 nodes communication among them increases, and with this their response time.

### 5. Conclusions

A parallel genetic algorithm with 2, 4 and 6 nodes in the cluster was developed for solving MKP. Each node starts with a different population, considering the bus schema for the migration between the slave nodes. PGA solved most of the test problems, since for four of them came at the best value reported. In the experiments, we can conclude that PGA showed better using 4-node cluster.

Operators used by the implemented genetic algorithms are crosses at one point, mutation and selection by default which offers GALib library. A major recommendation would perform the same experiments using various types of crossings offered by the library and so, increase the number of nodes used by the PGA.

We planned for further simplification of large size problems, to reduce the original problem size or to detect a priori whether or not a possible algorithm is a workable solution. It was noted that reduction for MKP can be achieved when there is no relationship between the coefficients of the objective function and the restrictions array.

In order to continue validating the proposed system, a generator of problems with a number of variables in a range of 5000-10000 is under work. For this generator, the PGA could obtain the solution using only four nodes of the cluster. It is clear that these results are not definitive, since the test problems do not present any kind of relationship between the system constraints coefficients, which is somehow what complicates the search of the overall optimum.

### 6. References


A Solution to Multidimensional Knapsack Problem using a Parallel Genetic Algorithm
Balderas_Posada Carlos, Vilariño_Ayala Darnes