A Parallel Genetic Algorithm to Speed Up the Resolution of the Algorithm Selection Problem

Alejandro Marrero
amarrerd@ull.edu.es
Departamento de Ingeniería Informática y de Sistemas,
Universidad de La Laguna
San Cristóbal de La Laguna
Tenerife, SPAIN

Eduardo Segredo
esegredo@ull.edu.es
Departamento de Ingeniería Informática y de Sistemas,
Universidad de La Laguna
San Cristóbal de La Laguna
Tenerife, SPAIN

Coromoto Leon
cleon@ull.edu.es
Departamento de Ingeniería Informática y de Sistemas,
Universidad de La Laguna
San Cristóbal de La Laguna
Tenerife, SPAIN

ABSTRACT

Deciding which optimisation technique to use for solving a particular optimisation problem is an arduous task that has been faced in the field of optimisation for decades. The above problem is known as the Algorithm Selection Problem (ASP). The optimisation techniques considered in previous works have been, mainly, approaches that can be executed rapidly. However, considering more sophisticated optimisation approaches for solving the ASP, such as Evolutionary Algorithms, drastically increases the computational cost. We are interested in solving the ASP by considering different configurations of a Genetic Algorithm (GA) applied to the well-known 0/1 Knapsack Problem (KNP). This involves the execution of a significant number of configurations of the GA, in order to evaluate their performance, when applied to a wide range of instances with different features of the KNP, which is a computationally expensive task. Therefore, the main aim of the current work is to provide, as first step for solving the ASP, an efficient parallel GA, which is able to attain competitive results, in terms of the optimal objective value, in a short amount of time. Computational results show that our approach is able to scale efficiently and considerably reduces the average elapsed time for solving KNP instances.

CCS CONCEPTS

• **Computing methodologies** → **Shared memory algorithms**.

GECCO '21 Companion, July 10–14, 2021, Lille, France

ACM ISBN 978-1-4503-8351-6/21/07…$15.00
https://doi.org/10.1145/3449726.3463160

KEYWORDS

Algorithm Selection Problem, Knapsack Problem, Genetic Algorithm, Parallelism

ACM Reference Format:
Alejandro Marrero, Eduardo Segredo, and Coromoto Leon. 2021. A Parallel Genetic Algorithm to Speed Up the Resolution of the Algorithm Selection Problem. In *2021 Genetic and Evolutionary Computation Conference Companion (GECCO '21 Companion), July 10–14, 2021, Lille, France.* ACM, New York, NY, USA, 4 pages. https://doi.org/10.1145/3449726.3463160

1 INTRODUCTION

The Algorithm Selection Problem (ASP) was proposed by Rice [7] in 1976 as a learning problem where the main goal is to learn the relationship between features from a set of instances of a particular optimisation problem and the performance of several optimisation algorithms when solving them. The overall performance on the set of instances can be maximised if the instances are correctly mapped to the best-performing algorithm considering the particular instance features.

The main is to develop a Meta Evolutionary Algorithm (MEA) for evolving or generating KNP instances, which are tailored to different configurations of an Evolutionary Algorithm (EA), particularly, a GA. Some of the most remarkable works found are the following.

Pospichal et. al [6], proposed a parallel GA for solving the KNP running entirely on a consumer-level of around $100 GPU. Besides, the authors defined several design criteria to optimise the performance over the GPU. In fact, the authors did not considered to include a repair operator in case the solutions were out of the feasible region of the search space. By contrast, the authors simply included a linear penalisation function whose value would be subtracted from the current fitness of the solution. Moreover, this penalisation value is associated with the situation where the capacity of the KNP is surpassed. Regarding the GA configuration and operators for solving the KNP instances, the authors set the

algorithm parameters as follows. First, the population size was set to 256 individuals for the quality test and from 2 to 256 increasing in powers of two for the speedup test. The number of items N for the KNP instances was quite small, considering only $N = 4, 20, 25, 40$ elements. In terms of genetic operators, the crossover operator was defined to the Uniform All Crossover [1] and the mutation operator to the Bit-Flip Mutation [1]. Besides, the selection operator was a parallel approach of the Tournament selection where two individuals were randomly chosen from the population. Lastly, the stopping criterion was 1,000 generations performed by the GA. Considering the results obtained by this implementation, compared to a traditional CPU parallel approach with GAlib,[1] the GPU solutions were slightly worse in terms of fitness than the CPU solutions for the same parameter. However, the average speedup of the GA GPU execution was 462 so the GPU could be utilised more time to improve the quality of the results.

Contrary to the previous work, Sonuc et. al [8] proposed a parallel approach for solving the KNP using a Simulated Annealing (SA) algorithm running on a GPU. This approach works as follows. First, the items introduced in the knapsack are sorted in descending order by density. The density of an item is calculated as the ratio between the profit and the weight of the item: $d_i = \frac{p_i}{w_i}$. It must be noticed that this term it is commonly known as efficiency. After that, the algorithm begins to introduce items in the knapsack until the maximum capacity of the knapsack is reached. Then, m SA algorithms are run in parallel using m different GPU threads. Even though this proposal is considerably easier to reproduce, the results cannot be extrapolated since the largest number of items considered for a KNP instance was set to 20.

In general, there are several approaches for solving the KNP on a GPU, however, most implementations consider the same type of genetic operators and parameters. Since we are interested in generating tailored KNP instances with a large variety of GA configurations, most of the GPU implementations are not suitable for this work. In other words, the GPU implementations are considerably faster than the parallel CPU approaches for solving the KNP. Nevertheless, creating several GPU GA implementations requires significantly more effort than producing the same configurations to run on a CPU. As a result, only parallel CPU approaches are considered herein.

2 PARALLEL GENETIC ALGORITHM

In spite of the large amount of EA techniques than can be applied to the KNP [1, 2, 4], this work is focused on designing a standard GA for solving the said problem. Specifically, the main objective here is to find an highly configurable GA implementation which is able to obtain high fitness values as fast as possible. The fitness value here considered is the objective function of the KNP, which is defined as follows:

$$max \quad \sum_{i=1}^{N} v_i x_i$$

$$subject\ to \quad \sum_{i=1}^{N} w_i x_i \leq Q \ and \ x \in \{0, 1\}$$

The number of elements that can be introduced in the knapsack is given by N, v and w are the profits and weights of the elements, respectively, and Q is the maximum capacity of the knapsack. Finally, the decision variables are represented by x.

The GA here presented (Algorithm 1) is similar to a standard GA [1]. However, the main difference lies on the replacement strategy performed at the end of each generation of the approach, which has been designed having in mind its parallelisation. The algorithm starts by creating an initial population of random solutions. After that, and while the stopping criterion is not met, the algorithm performs the following steps. For each individual, two different individuals are selected by using a Binary Tournament Selection operator [1]. The selected individuals, known as parents, will undergo the reproduction or mating phase, where new offspring will be created. The reproduction phase consist of applying the genetic operators of the standard GA, i.e., crossover and mutation. The particular genetic operators chosen for this work are the Uniform Crossover (UC) and the Uniform One Mutation (UOM) [1]. Afterwards, the new offspring are evaluated using the KNP objective function. Finally, the replacement approach is based on a First Improve version between each individual jth in the population and the jth offspring. Algorithm 1 is easily parallelised since the individuals of the population can be mapped to different cores to be evaluated and replaced in parallel. Concretely, Algorithm 1 performs the main loop, between lines 3 and 9, in a parallel way. The parallelisation approach consists of dividing the population of individuals among the total number of cores available to perform the computation. Therefore, each core only works on a portion of the population (PS/n_cores).

3 EXPERIMENTAL ASSESSMENT

In order to evaluate the performance of the aforementioned parallel GA, a two-phase experimental evaluation was conducted. First of all, different configurations of the GA proposed were evaluated by performing sequential runs for solving a set of KNP instances with different features and a variable number of items N equal to 50, 100, 500 and 1,000. The types of the KNP instances considered were some of

[1] GAlib: http://lancet.mit.edu/ga/

Algorithm 1: Genetic Algorithm

```
Data: PS, N, V, W, Q
Result: Population of solutions for a KNP instance
population = initialisation(PS);
while not stopping criterion reached do
    for j ← 0 to PS do
        firstParent = selection(population);
        secondParent = selection(population);
        offspring_j = reproduction(firstParent,
          secondParent);
        evaluation(offspring_j);
        population_j = replacement(population_j,
          offspring_j);
    end
end
```

those proposed by Pisinger [5]. Concretely, in this work, the Uncorrelated, Inverse Correlated, Strongly Correlated, Subset Sum and Spanner Strongly Correlated instances were considered. After that, in a second experiment, one of the best-performing configurations, in terms of fitness, considering the biggest KNP instances was selected to assess its scalability across multiple cores in several parallel runs.

Experiments were performed using a machine with two AMD Opteron processors (model 6164) with 48 cores running a Debian distribution (Buster). In addition, the problem and all the GA implementations were written in C++.

3.1 First experiment: parameter setting analysis

The first experiment was aimed to analyse the best GA configuration for each type of the KNP instances considered. At this point, it is worth mentioning that the GA was run sequentially. For this purpose, 16 different configurations were designed by varying the values of different parameters. The population size was set to 40, 80, 120 and 160 individuals. At the same time, the mutation rate (*mr*) was set considering the size of the KNP instance to solve. Therefore, it was defined as $1/N$, were N is the number of items or size of the KNP instance. The crossover rate (*cr*) was set to values 0.7, 0.8, 0.9 and 1.0. Last but not least, the stopping criterion was set to $3 \cdot 10^5$ evaluations and each execution was repeated 30 times, since we are dealing with stochastic approaches.

Following the evaluation procedure performed in [3], the *fitness* was the metric selected to compare the different GA configurations. So as to give the conclusions with enough statistical confidence, *Shapiro-Wilk, Levene, ANOVA or Welch* statistical tests were considered for results that follow a normal distribution, while the *Kruskal-Wallis* test was applied otherwise. Configuration A statistically outperforms configuration B if the p-value obtained after performing a pairwise comparison of both approaches by following the statistical procedure described before is lower than the significance level $\alpha = 0.05$ and, if at the same time, configuration A provides, at the end of the runs, a higher mean and median of the fitness than configuration B.

Observing the results[2], we note that, considering a particular type of instance, statistically significant differences arose among different configurations of the GA. The above means that the parameterisation of the GA has a direct impact over its performance.

Furthermore, changing the size of the instances could modify the behaviour of the different GA configurations. The above means that, depending on the particular features of an instance, the best-performing GA configuration changes. The fact that different GA configurations are much more suitable for different types of KNP instances demonstrates the importance of properly solving the ASP in this particular context. Given a KNP instance with a particular set of features, we are interested in providing a suitable GA configuration that attains high quality solutions for the mentioned instance.

Finally, by the application of the future MEA, we could generate KNP instances that intentionally fits to a given GA configuration, i.e., that can be solved properly by the target GA configuration we are interested in. The above would allow large data sets of KNP instances to be obtained, for which we would know the best-performing GA configuration for each of them.

3.2 Second experiment: scalability analysis

The main goal of the second experiment was to study whether the parallel GA is able to scale properly, i.e., if it is able to provide results in a shorter amount of time when the number of computational resources to run it is increased. The reader should recall that this parallel GA will be used as a component of the future MEA that will generate tailored KNP instances. Since the above task will consume a huge quantity of computational results, it is very important that the parallel GA scales properly, thus speeding up the whole procedure.

Although previous results showed that there is not a silver bullet configuration that obtains the best results in every scenario, for the sake of saving computational time, the scalability study was performed with only one of the 16 GA configurations defined in the first experiment. For the above

[2]Due to the length restrictions of this paper, the whole set of tables, results, source code and additional related information is available in a Github repository https://github.com/PAL-ULL/GECCO-21-Parallel-GA-KNP

reason, that GA configuration obtaining, the largest number of times, the first position in the ranking was selected for the current experiment. Particularly, the said configuration applied 160 individuals and a crossover rate $cr = 1$. Furthermore, only instances of size N = 1,000 items, i.e., the biggest ones, were taken into consideration. In an effort to efficiently distribute the workload, the different number of cores were selected to be divisors of the number of individuals in the population. As a result, considering a population size of 160 individuals, the number of cores contemplated in the scalability analysis were 2, 4, 5, 8, 10, 16, 20, 32 and 40.

Figure 1 shows the speed up results obtained during the scalability study for a Spanner Strongly Correlated instance of N = 1,000 items. Although Figure 1 shows that the speed up is not completely linear, results demonstrate that the parallel GA can scale efficiently when increasing the number of cores. For instance, the average elapsed time for solving this particular KNP instance sequentially, considering 30 repetitions, was equal to 14.1661 seconds, while considering 40 cores, the average elapsed time decreased to 0.8525 seconds, thus resulting in a speed up factor over 16.

4 CONCLUSIONS AND FUTURE LINES OF WORK

The main objective of this work is to propose a parallel GA for solving KNP instances which is highly configurable and can scale efficiently over several cores. The results obtained in the experimental assessment support the hypothesis that there is not a silver bullet GA configuration for solving all types of KNP instances, thus demonstrating the importance of properly addressing the ASP in this particular context. The generation of tailored KNP instances for a particular GA configuration involves a huge amount of computational resources and, as a result, to speed up the whole procedure is mandatory. The scalability analysis performed over the parallel GA proposed herein demonstrates that it is able to properly accelerate the resolution of different types of KNP instances.

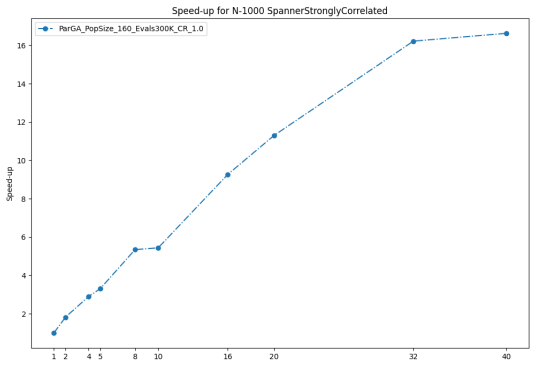

Figure 1: Speed up values considering the Spanner Strongly Correlated KNP instance of N = 1,000 items

Bearing the above in mind, one of the main lines of future work will involve the development of an MEA that makes use of the parallel GA here proposed herein in order to decrease the time required to generate tailored KNP instances. Once we get sufficiently large data sets of KNP instances that fit with different GA configurations, we will be able to train ML models that allow the resolution of the ASP, in this particular context, to be carried out in an easier manner.

ACKNOWLEDGEMENT

This work was partially funded by the Spanish Ministry of Science, Innovation and Universities, as well as by the University of La Laguna, as part of the programme "Nuevos Proyectos de Investigación: Iniciación a la Actividad Investigadora" [contract number 1203_2020]. The work of Alejandro Marrero was funded by the Canary Islands Government "Agencia Canaria de Investigación Innovación y Sociedad de la Información - ACIISI" [contract number TESIS2020010005].

REFERENCES

[1] Agoston E. Eiben and James E. Smith. 2003. *Introduction to Evolutionary Computing*. Springer, Berlin Heidelberg. 299 pages. https://doi.org/10.1162/evco.2004.12.2.269 arXiv:9809069v1 [arXiv:gr-qc]

[2] Kangshun Li, Yuzhen Jia, Wensheng Zhang, and Yang Xie. 2008. A new method for solving 0/1 knapsack problem based on evolutionary algorithm with schema replaced. In *2008 IEEE International Conference on Automation and Logistics*. IEEE, Qingdao, China, 2569–2571.

[3] Alejandro Marrero, Eduardo Segredo, Coromoto León, and Carlos Segura. 2020. A Memetic Decomposition-Based Multi-Objective Evolutionary Algorithm Applied to a Constrained Menu Planning Problem. *Mathematics* 8, 11 (2020). https://doi.org/10.3390/math8111960

[4] Phuong Hoai Nguyen, Dong Wang, and Tung Khac Truong. 2016. A new hybrid particle swarm optimization and greedy for 0-1 knapsack problem. *Indonesian Journal of Electrical Engineering and Computer Science* 1, 3 (2016), 411–418. https://doi.org/10.11591/ijeecs.v1.i3.pp411-418

[5] David Pisinger. 2005. Where are the hard knapsack problems? *Computers & Operations Research* 32, 9 (2005), 2271–2284. https://doi.org/10.1016/j.cor.2004.03.002

[6] Petr Pospichal, Josef Schwarz, and Jiri Jaros. 2010. Parallel Genetic Algorithm Solving 0/1 Knapsack Problem Running on the GPU.

[7] John R. Rice. 1976. The Algorithm Selection Problem. *Advances in Computers* 15, C (1976), 65–118. https://doi.org/10.1016/S0065-2458(08)60520-3

[8] Emrullah Sonuç, Baha Sen, Emrullah Sonuc, Baha Sen, and Safak Bayir. 2016. A parallel approach for solving 0/1 knapsack problem using simulated annealing algorithm on CUDA platform Keratoconus Disease and Three-Dimensional Simulation of the Cornea Throughout The Process Of Cross-Linking Treatment View project A Parallel Approach.