Algorithm Configuration using GPU-based Metaheuristics

Roberto Ugolotti, Youssef S.G. Nashed, Pablo Mesejo, Stefano Cagnoni
Department of Information Engineering, University of Parma, Italy
{rob_ugo, nashed, pmesejo, cagnoni}@ce.unipr.it

ABSTRACT

In this paper, a GPU-based implementation of Differential Evolution (DE) and Particle Swarm Optimization (PSO) in CUDA is used to automatically tune the parameters of PSO.

The parameters were tuned over a set of 8 problems and then tested over 20 problems to assess the generalization ability of the tuners. We compare the results obtained using such parameters with the 'standard' ones proposed in the literature and the ones obtained by state-of-the-art tuning methods (irace). The results are comparable to the ones suggested for the standard version of PSO (SPSO), and the ones obtained by irace, while the GPU implementation makes tuning time acceptable.

To the best of our knowledge, this is the first time that a general purpose library of GPU-based metaheuristics is used to solve this problem, as well as being one of the few cases where DE and PSO are both used as tuners.

Categories and Subject Descriptors
I.2.8 [Artificial Intelligence]: Problem Solving, Control Methods, and Search—Heuristic methods

Keywords
Parallel Computing; GPGPU programming; Automatic Parameter Configuration; Evolutionary Algorithm; Swarm Intelligence

1. INTRODUCTION

One of the main problems to be tackled when applying Evolutionary Algorithms (EAs) and Swarm Intelligence (SI) techniques is the strong influence of parameter selection on their performance.

In this paper, we use the GPU implementation of two real-valued population-based optimization techniques (Particle Swarm Optimization (PSO) and Differential Evolution (DE)) to automatically configure another metaheuristic (PSO). This SI method is tuned using a benchmark of 8 different optimization problems (also implemented on GPU).

The main contribution of this work is three-fold: (i) the first documented use, to the best of our knowledge, of GPU-based metaheuristics devoted to the automatic configuration of algorithms, (ii) a comparison with the results obtained using the parameters selected by the irace software package, and using a classically well-established set of manually selected parameters, and (iii) the use of metaheuristics not only to tune numerical parameters but also to make some qualitative decision about the design of the algorithm to tune, by choosing categorical/nominal parameters (as, for instance, the PSO topology).

2. PROPOSED METHOD

The implementation of the metaheuristics employed in this paper has been presented in [5]. The code used in this research is available at http://sourceforge.net/p/libcudaoptimize.

When DE or PSO are used as tuner, they work as any other optimizer. The only difference lies in fitness evaluation and how fitness values are stored and used. Each particle in a tuner represents a set of parameters of the optimizer to tune (in this case PSO). This optimizer is tested T times on a set of F functions. The resulting fitness is then computed based on the average and standard deviation of the best fitness obtained over all functions; that is a 2 x F array of values. A set of parameters is better than another if it obtains better results in the majority of the F functions. In case of a tie, the winner is selected comparing the sum of Welch’s t test values over the N functions under consideration:

\[ R = \frac{1}{N} \sum_{i=1}^{N} \frac{(\bar{X}_a - \bar{X}_b)}{\sqrt{\frac{s_a^2}{n_a} + \frac{s_b^2}{n_b}}} \]

where a and b are the two sets of parameters (or the two optimizers), \(\bar{X}_a\) and \(\bar{X}_b\) are the mean and standard deviation obtained by each optimizer on function i. If R < 0, the optimizing a is better (since we are looking for the minima of the functions), otherwise b wins. During the tuners’ execution, optimizers are compared pairwise while, at the end of the optimization process, the surviving particles are compared with one another in a full tournament to choose the winner.

PSO and DE are real-valued optimization methods, so the problem of representing nominal parameters needs to be addressed. Our choice has been to represent each nominal value by a vector with as many values as the options available and choose the parameter corresponding to the element with the highest value. For instance, one of the parameters we want to optimize in PSO is topology. We can

Figure 1: Representation of a set of PSO parameters inside the tuner. The first four elements represent numerical parameters (either integer or real-valued numbers), while the last three contribute to the decision about a nominal parameter (in this case topology).
choose between ring, star and “global-best” topology. This means that three values for each parameter in the PSO/DE tuner are reserved to this choice and the largest one in the best particle is chosen as winner (see Figure 1). The range of the PSO parameters allowed by the tuning algorithm are \( C_1, C_2, C_3 \), where \( C_1 \in [0, 4], C_2 \in [0, 1,5], C_3 \in [0, 10], \) and topology is \( \{ \text{gbest, Ring, Star} \}. \) The best automatically-tuned parameter values found by DE, PSO, and \textit{irace}, in 10 runs of every tuner, and 10 of PSO with standard parameters. See [5] for a definition of the symbols.

<table>
<thead>
<tr>
<th>Param.</th>
<th>DE</th>
<th>PSO</th>
<th>irace</th>
<th>SPPO</th>
</tr>
</thead>
<tbody>
<tr>
<td>( C_1 )</td>
<td>0.953</td>
<td>1.525</td>
<td>1.721</td>
<td>1.996</td>
</tr>
<tr>
<td>( w )</td>
<td>0.636</td>
<td>0.443</td>
<td>0.351</td>
<td>0.730</td>
</tr>
<tr>
<td>Pop. Size</td>
<td>37</td>
<td>34</td>
<td>35</td>
<td>60</td>
</tr>
<tr>
<td>Topology</td>
<td>gbest</td>
<td>gbest</td>
<td>gbest</td>
<td>gbest</td>
</tr>
</tbody>
</table>

### 3. RESULTS AND CONCLUSIONS

We performed parameter tuning on a training set of 8 functions (numbers 0, 5, 6, 8, 9, 10, 12, 13 in [5]) and a validation set of 20. Our goal was to check (i) whether our method was able to get results competitive with other more complex and/or computationally expensive state-of-the-art tuners (\textit{irace}). (ii) whether we could find repeatable patterns in the parameter sets obtained by different techniques, and (iii) the generalization ability of every tuner.

The comparison was performed by running 10 times our tuners and \textit{irace}, and searching the best population size and topology with the parameter values for \( C_1, C_2, w \) suggested in [5]. We then selected the best parameter configuration found by each method, based on the Friedman test and the visual examination of the results using boxplots. The resulting configurations, shown in Table 2, are similar: all of them use global best topology, a small population size while the setting of parameters \( C_1, C_2 \) and \( w \) lie within very small ranges. It should be noticed that the three tuned optimizers reach good results also in functions that were not included in the training phase, showing good generalization.

In conclusion, we can say that simple EA-based parameter tuning is able to get comparable results to a well-established state-of-the-art method in parameter tuning, like \textit{irace}. Moreover, in our case, the parallel implementation makes the tuning process computationally efficient, which allows our tuners to perform their task in a much more reasonable time with respect to sequential implementations.

### Acknowledgments

Roberto Ugolotti is funded by Compagnia di S.Paolo and Fondazione Cariparma. Youssef S. G. Nashed, and Pablo Mesejo are funded by the European Commission (Marie Curie ITN MIBISOC, FP7 PEOPLE-ITN-2008, GA n. 238819). The authors want to thank Marco Grossi and Gian Maria Marconi for their precious help.

### 4. REFERENCES


