Proceedings of Engineering & Technology (PET)
pp. 127-131

2^{nd} International Conference on Automation, Control, Engineering and Computer Science (ACECS-2015)
Proceedings of Engineering & Technology (PET)

A web-based genetic solver for permutation flowshop using the Design Pattern

Lamia Trabelsi
Ecole Suprieure des Sciences Economiques
et Commercial de Tunis (ESSECT), Tunisie
Email: lamia_tr2001@yahoo.fr

Talel Ladhari
College of Business, Umm Al-Qura University,
Umm Al-Qura, Saudi Arabia
Ecole Suprieure des Sciences Economiques
et Commercial de Tunis (ESSECT), Tunisie.
Email: talel_ladhari2004@yahoo.fr

***Abstract*—In this paper, we present a genetic solver tool to solve permutation flowshop problem. To implement the solver's functionalities, we introduce the use of design patterns such as the composite, template and observer patterns to define collaboration between the genetic operators. The proposed genetic solver will evaluate several versions of genetic algorithm, and then, give the best version of the selected flowshop variant using the student test. Moreover, it allows researchers to download the executable code of the best generated version of the genetic algorithm and the documentation of the selected flowshop variant.**

I. INTRODUCTION

Combinatorial optimization problems under permutation property (COP-PP) are generally NP-hard. To address such problems, researchers and practioners grew more attracted to approximate approaches. Because of the growing interest in such methods, in the last decade, several ready-to-use tools have been proposed in the literature such as HeuristiticLAB [1], E-OCEA [2], LEKIN [3] and LISA [4]. It is widely admitted that ready-to-use tools in optimization help to improve the quality of searches. However, most of them are not available like E-OCEA [2]. Moreover, heuristics calibration tools have not been supported by most studied cases. In [5], we have proposed a new Research Support System for COP-PP. The new RSS-COPP provides tools to help young researchers to select a COP-PP variant, to solve the studied problem by a metaheuristic algorithm and to generate the best metaheuristic configuration leading to the best solution using calibration tools. The Genetic solver is the core of RSS-COPP. In this paper, we propose to enhance the functionalities of the genetic solver for permutation flowshop by introducing the use of the design patterns and the generation of the executable code of the best configuration. In section II, we present the used design patterns to deal with collaboration between the several genetic operators. In section III, we present our new genetic algorithm solver. Section IV concludes and proposes future research avenues.

II. INTEGRATING THE USE OF DESIGN PATTERNS IN THE GENETIC SOLVER

We propose to use design patterns to enhance collaboration among our genetic solver's operators' components. The used patterns combine the composite and template patterns to deal with the operator structure and the observer pattern to deal with the population's update phase.

A. Operator research structure

Empirically speaking, experimental studies showed that metaheuristics performance depends on the use of some different operators and their behaviour. In fact, effectiveness of metaheuristics depends on how to use the different available operators. In this section, we address the problem of collaboration between metaheuristics and their operators by presenting an operator structure. In fact, we believe that two issues make up the collaboration problem that should be addressed when developing our genetic solver:

- *How to call an operator:* in most cases, operators are invoked into a metaheuristic algorithm under certain conditions to deal with the diversification/ intensification paradigm. This can take place, for example, when we apply a local search, a crossover in evolutionary algorithms or a re-construction of metaheuristic memories.
- *The operator choice*: in fact, under some conditions, the selection of one operator from an available list of operators at run-time may improve the effectiveness of the designed metaheuristic.

a) The operator structure description: Our proposed operator structure rests on the combination of two patterns; a Template pattern and a Composite pattern [6]. The use of such patterns is justified by several reasons. In most cases, calling research operators in metaheuristics follows a common behaviour as invoked in our research problem statement. If conditions and constraints are satisfied, operators will execute their main operation or else they will execute another operation. For this reason, we propose to use the template pattern. In fact, the template pattern is the most suitable pattern to describe this common skeleton, letting subclasses redefine some operational steps. However, before executing an operator operation, a metaheuristic is faced with the problem of the choice of the operator to be invoked. To deal with the operator choice issue, we combine the Template Pattern with the Composite Pattern for the following reasons:

- The Composite pattern allows operators to be grouped to achieve the same treatment in a given context (metaheuristic).
- The Composite pattern allows for treating the composite operator (operator list) and the simple operator uniformly. In this way, the Composite pattern hides the complexity of the operator's structure from the

ISSN: 2356-5608

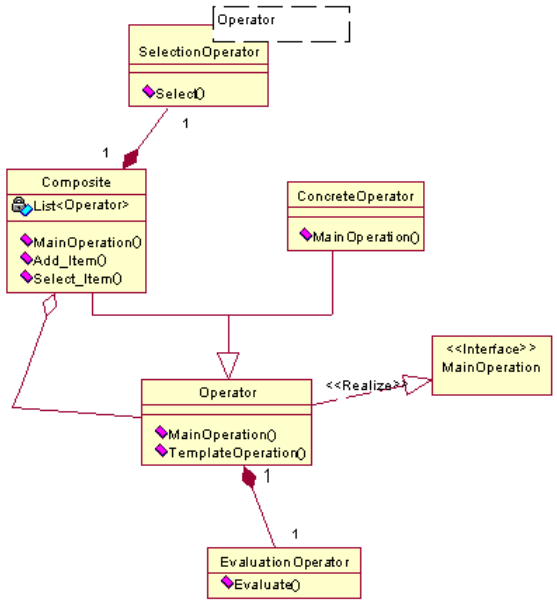

Fig. 1. Generic Operator structure

```
TemplateOperation(){
if(evaluationOperator.Evaluate()){
this.MainOperation();
}
}
```

Fig. 2. PseudoCode of the Template Method Construct()

metaheuristic. Metaheuristics can use a single operator or an operator selected from a list by applying a selection operator in the same way. Moreover, adding a new operator or subtracting an existing one is easier and it does not affect the functioning of the metaheuristic.

The proposed structure is shown in figure 1. The main participants are:

- Operator: it is an abstract class which implements the generic template TemplateOperation() (see pseudocode in figure 2).
- Main_Operation Interface: it presents the Main_Operation method to be implemented in concrete operators
- ConcreteOperator: the ConcreteOperator is the specialization class of Operator.
- Composite: Composite is a subclass of Operator and it consists of a list of Operator classes(Leaf). If we proceed to select one operator, the composite main_operation is implemented by the following steps(figure 3): the composite selects an item according to a selectionOperator class. The Selected Item will call in its proper main_operation method ().

```
MainOperation(){
item=Select_Item()
item.MainOperation();
}
```

Fig. 3. PseudoCode of the Composite Main_Operation example

b) Consequences: In summary, the proposed operator will benefit from all the advantages of the combined patterns that we mentioned above. These mainly include:

- Hide complexity: it treats single or composite operators uniformly.
- Flexibility: Adding new operators or subtracting an existing one is much easier at run-time.
- A Dynamic metaheuristic: The Metaheuristic will select the most suitable operator for the situation at run-time thanks to the use of operators list in the CompositeOperator. In the original Composite pattern, all Leaves (concrete operators) will be executed. This can be applied when we need all operators to be executed. For example, to construct a population, we use a heuristic constructive operator and a random operator. However, when only one operator must be invoked, the composite operator will be aggregated to a selection operator class to select a given operator.

B. The observer pattern for the population update phase

In genetic algorithm, the best solution and the number of generation are the observers of the population class. When the population is updated, these classes will be notified about this update and will proceed by updating their values. As shown in 4, Population, as a specialized class of the subject class, implements the Attach() operation. In this operation, we will add the observers to the observers list. The Bestsolution and IterationNumber classes realize the observer interface. By the notify() operation, the population notifies its relative observers about its update. Similarly, we have noted that Max_With_Out_Improvement is an observer of the best solution class. When the best solution is updated, Max_With_Out_Improvement will assess the best solution value. If the best value has changed, Max_With_Out_Improvement sets its value at zero, otherwise it will be incremented. In this way, the Best solution inherits from the subject class its operation and Max_With_Out_Improvement will realize the observer interface.

III. The Genetic-Solver foundation

The genetic solver has been implemented as a first application for permutation flowshop. More detail about its foundation is found in [5]. In the next section, we will present the main genetic solver interfaces.

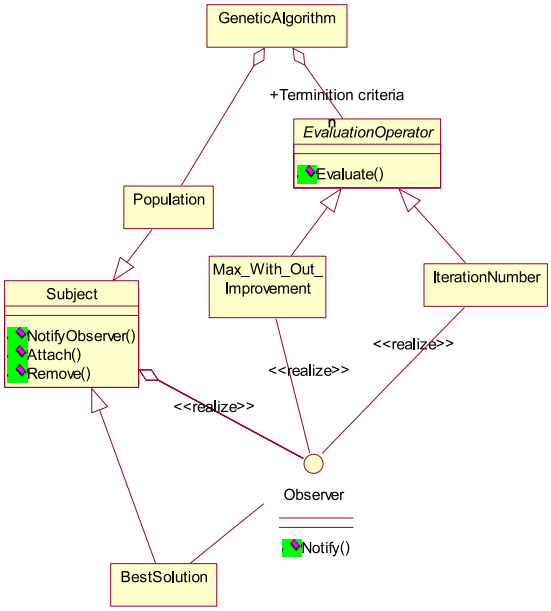

Fig. 4. The proposed observer pattern

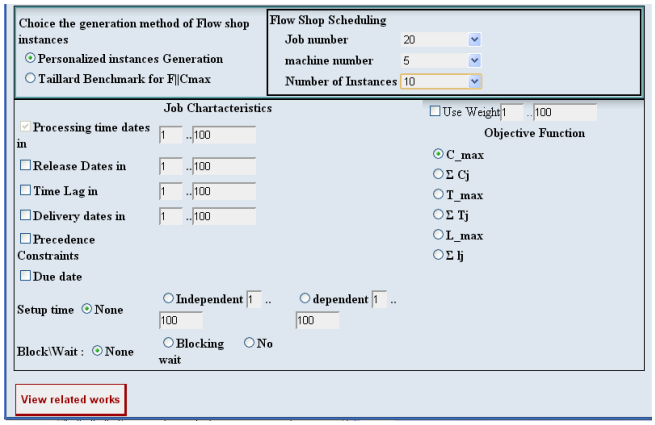

Fig. 5. The problem setting interface

A. Selection of a permutation flowshop variant

It is the first step in our solver. It provides an interface to select a variant for a permutation flowshop scheduling problem and to specify its constraints. It allows for the choice of the generation procedures of the tested problems or instances (figure 5).

B. Metaheuristic parametrization

After the selection of the to-be-solved variant, this step allows the researcher to select the to-be-tested metaheuristic's parameters values. Figure 6 presents the initialization phase which allows the user to define the relevant population size and the used operator to construct the population.

C. Result Generation and code downloading

When the different metaheuristic parameters have been selected, the solver proceeds to the solving process and to finding the best result. The best result will be shown in the result page (see figure 7) The result interface will provide

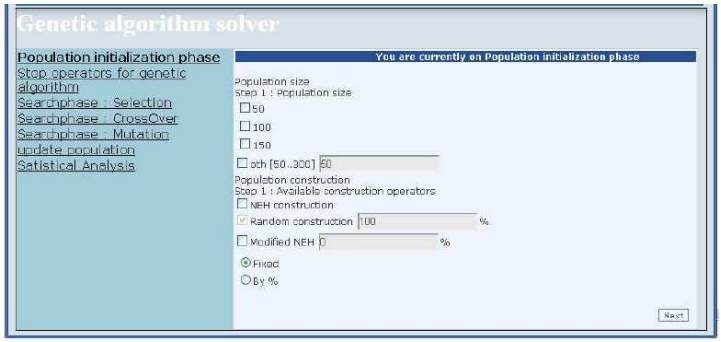

Fig. 6. Initialization phase

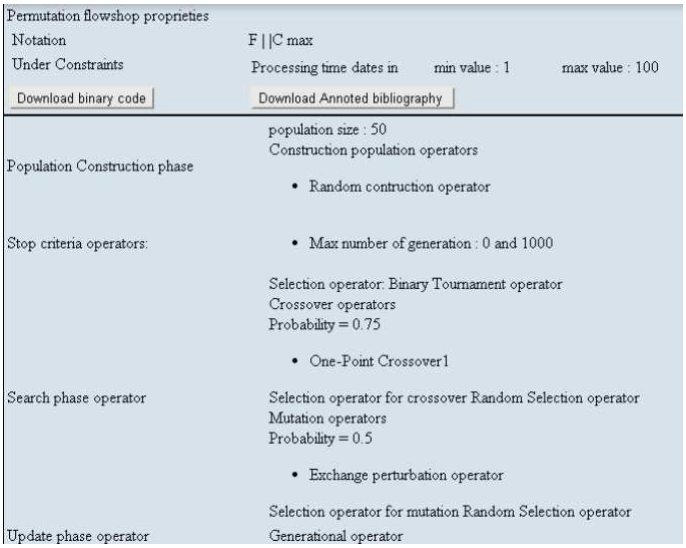

Fig. 7. Example of a result page

an opportunity to download the different generated files (the binary code and the annotated bibliography). Figure 8 shows the resulting binary code of the best generated configuration and the execution of the file on the researcher side.

IV. Conclusion and Future Direction

The Genetic Solver tool is a web-based application developed in order to help young researcher to find the best version of a genetic algorithm for a COP-PP problem. As a first application to permutation flowshop, the proposed tool provides an interface for the selection of a flowshop problem variant, an interface to choose the value of genetic parameters, an interface to generate the best genetic version result and the possibility to download the executable code of the relative version. The operator component is enhanced by combining composite and template patterns. Essentially, the new structure allowed us to manipulate operators uniformly and to hide their complexity. The Solver should integrate more sophisticated and more powerful statistical data analysis such as ANOVA [7] to return the best reliable parameters values of a selected metaheuristic. In addition, tools such as fitness landscape

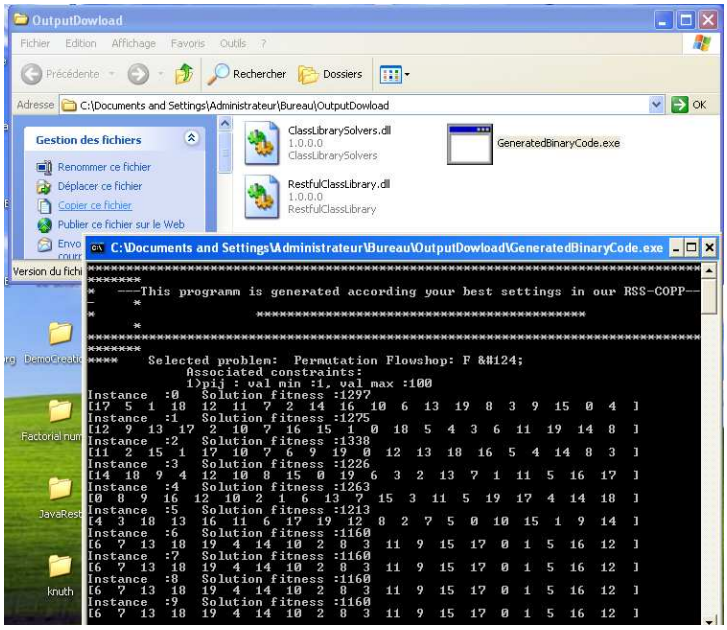

Fig. 8. Example of an execution of the binary code in the researcher side

and robustness analysis will be proposed for the permutation flowshop.

REFERENCES

[1] S. Wagner, A. Beham, G. Kronberger, M. Kommenda, E. Pitzer, M. Kofler, S. Vonolfen, S. Winkler, V. Dorfer, and M. Affenzeller, "Heuristiclab 3.3: A unified approach to metaheuristic optimization," *Proceedings of the VII Congreso Espanol sobre Metaheursticas, Algoritmos Evolutivos y Bioinspirados (MAEB 2010), Valencia, Spain.*, 2010.

[2] V. Tḱindt, J.-C. Billaut, J.-L. Bouquard, C. Lente, P. Martineau, E. Nron, C. Proust, and C. Tacquard, "The e-ocea project: towards an internet decision system for scheduling problems," *Decision Support Systems*, vol. 40, pp. 329–337, 2005.

[3] N. Asadathorn, "Scheduling of assembly type of manufacturing systems: Algorithms and systems developments," Ph.D. dissertation, Department of Industrial and Manufacturing Engineering, New Jersey Institute of Technology, Newark (USA),, 1997.

[4] H. Braesel and N. Shakhlevich, "Lisa–fit for cooperative development,," *Sixth Workshop on Models and Algorithms for Planning and Scheduling Problems (MAPSP'03)*, pp. 107 – 108, 2003.

[5] T. Lamia and L. Talel, "A new web-based solver for combinatorial optimization problems under permutation property," in *Modeling, Simulation and Applied Optimization ICMSAO13 (28-30 april 2013, Hammamet)*, 2013.

[6] E. Gamma, R. Helm, and R. Johnson, *Design Patterns. Elements of Reusable Object-Oriented Software.*, 1st ed. Addison-Wesley Longman, 1995.

[7] D. Montegromy, *Design and Analysis of Experiments*, 5th ed. John Wiley and Sons Inc, 2001.