*Multidisciplinary Informatics Journal*
*Vol. 1, No. 1, June 2018, Pg. 17-26*

# SCRIPTING LANGUAGE FRAMEWORK IN SOLVING FACILITY LAYOUT PROBLEM (FLP) BASED ON META-HEURISTICS HYBRIDIZATION

S.Masrom[1,*], N.Ahmad[1] , N.Omar[2], and A.S.A. Rahman[3]

[1]Faculty of Computer and Mathematical Sciences,Universiti Teknologi MARA, Perak Branch Tapah Campus, Perak, MALAYSIA

[2]Faculty of Computer and Mathematical Sciences,Universiti Teknologi MARA, Shah Alam, Selangor, MALAYSIA

[3]Faculty of Science and Information Technology, Universiti Teknologi PETRONAS, Perak, MALAYSIA

Author Correspondence, e-mail: suray078@perak.uitm.edu.my

Received: 10 April 2018 / Accepted: 15 May 2018/ Published online: 1 June 2018

**ABSTRACT**

Facility layout can be seen everywhere in people's daily life and industry so that it has a broad research background, which returns the economic and social benefits. Besides time critical, another challenge of creating a facility layout is to resolve dynamic factors, such as the modification of building layout and the placement of new elements in the layout. Users may rely on computerized facility layout, but when involves with dynamic factors, a great deal of technical hurdles appears mainly in the computer programming. Therefore, this research proposes an easy programming approach to solve facility layout problem with meta-heuristics algorithm in a scripting language software framework. Evaluation results indicated that the codes of the scripting language are simpler and concisely describe the algorithms in a directly publishable form.

**Keywords:** scripting language; facility layout; optimization; meta-heuristics.

## 1. INTRODUCTION

The Facility layout problem (FLP)[1] exists everywhere in people's daily life, organizations and industries. The FLP is a well-studied combinatorial optimization problem that emerged in a variety of problems such as layout design of hospitals, schools, airports, networking and backboard wiring. The most common objective in FLP is minimizing the facility resources costs that are determined based on the flow between the facilities and the distance between each facilities location. Due to the dynamic and impulsive environment in today's industry operations, dynamic FLP appears to be very important. The dynamic FLP extends the static FLP by involving the changes in resources flow over multiple periods as well as the costs of rearranging the layout. Users may rely on computerized facility layout, but when involves with dynamic factors, a great deal of technical hurdles appears mainly in the computer programming. The objectives of this paper are two-folds. First is to demonstrate an easy and a flexible of scripting language for developing algorithms in solving the facility layout optimization problem. Second objective is to incorporate two meta-heuristics algorithms namely Particle Swarm Optimization(PSO) and Genetic Algorithm(GA) for solving the FLP. While scripting language has been widely used for many kinds of complex systems, very little scientific study has been reported about scripting language for the facility layout optimization problem based on PSO-GA hybrids.

This paper is organized as follows. Section 2 describes the research background of FLP, the meta-heuristics approach and about scripting and programming languages. Section 3 describes the proposed scripting language of the PSO-GA hybrids for FLP followed by the evaluation results in Section 4. Section 5 presents the conclusion of this paper.

## 2. BACKGROUND OF STUDIES

### 2.1. Definition of FLP

The formulated models of the FLP can be adopted based on Quadratic Assignment Model (QAP), Mix Integer Programming (MIP) and Graph Theory. QAP [1] model is considered as NP-complete and the objective is to assign all facilities to different locations with the goal of minimizing either the sum of the distances, the time and/or cost for reaching the locations multiplied with the corresponding flows. Graph theory [2] model is defined as a node within a graph network that rely on a predefined desirable adjacency of each pair of facilities . In other words, the graph theory approaches, it is assumed that the desirability of locating each pair of facilities adjacent to each other is known. MIP model also has gained some attention as a way of modeling the FLP such as in the research done by [3]. Interest ideas were reported in the paper is a combination of MIP with Genetic Algorithm, which is a common meta-heuristics algorithm.

### 2.2. Meta-heuristics approach in solving FLP

Various meta-heuristics such as Particle Swarm Optimization and Genetic Algorithms have been used to approximate the solution of FLP. Currently, research that used GA for FLP in solving dynamic properties of the problem has been reported in[4][5][6]. Some approaches that used meta-heuristics hybridizations also found to be effective[7][8]. Majority of these research used meta-heuristics as to finding the minimum total distances for reaching all the input locations. Since most researchers working with meta-heuristics are required to develop their own algorithm for their specific problem including FLP, the task of selecting, modifying and extending codes from reusable software is very likely to occur. The following part describe the programming approaches for meta-heuristics algorithms development.

### 2.3. Programming Approaches for Meta-heuristics

To support easy programming, the use of existing software to build a new software by reusing or modifying the provided codes has been a common practice in software[9]. To date, there exist many kind of software tools that support codes and generally, the two common programming approaches provided by the software are graphical user interfaces (GUI) or text programming language. Each of these provides a functionality that a programmer can use for their own purposes, but in slightly different forms and each is associated with benefits and drawbacks.

GUI is a programming approach operated at the front-end of reusable software, which performs operations in a drag-drop programming environment. GUI is very easy and convenient, but it is clumsy and has a strict rigidity to pre-defined functions of a software library or framework [10]. For meta-heuristiscs based application, usually GUI is used as a medium interface for algorithm and experimental configurations, or to display optimization results with visuals such as graphs and charts, for examples in rapid software like HotFrame [11], iOpt[12], HeuristicLab[13], JCLEC [14] and Opt4J[15]. Researchers in [16] provides an extensive reviews of the meta-heuristics software tools and clearly highlight about the limitation of GUI programming approach to support easy programming for meta-heuristics mainly for hybridization of different algorithms.

Furthermore, text programming language like JAVA and C++ have more flexibility than GUI for software extension. Once a developer is familiar with the programming language used in reusable software, they can be very productive for a variety of software creations. Text programming language are known as a highly efficient language and usually being used for developing back-end software libraries or software frameworks. Nevertheless, programmers must have a good programming skill as text programming languages are more difficult than GUI programming approach. The programs can be very lengthy, which disadvantage users to directly used the programs for publication or documentation. Alongside this, GUI is limited in providing the precise textual codes. As discussed previously, to convey information with GUI sometimes demands more pages than textual codes. When considering to support both easy and concise programming, scripting based programming language should be considered as the best programming platform. Scripting language is a text-based programming but is simpler and has lesser codes than JAVA and C++.

## 3. THE PROPOSED SCRIPTING LANGUAGE

The scripting language proposed in this paper was an extension from the JACIE (Java-based Authoring Language for Collaborative Interactive Environments) [17] scripting language software framework. The JACIE consists of an intermediate compiler that allow new language specifications to be constructed. The compiler translates the propose scripting language into another JAVA codes for the execution. The enhancements have been done to allow the inclusions of new scripting language constructions for meta-heuristics, the hybridization and FLP. Details about the JACIE enhancement for the meta-heuristics hybridization that used PSO-GA is described in our previous paper [16]. The overall software architecture is divided into front-end scripting language, intermediate compiler and the back-end software framework. The scripting language will be converted into JAVA codes by the JACIE compiler. These JAVA codes can be implemented in the JSwarm [18] software framework. The proposed scripting language described in the paper is divided into five components namely *general, PSO Update, GA crossover, GA mutation* and *Dynamic parameterizations.* One of the main element in the general component is problem specification. Figure 1 presents the syntactic specifications of the general component with problem specification for FLP.

```
SGMutation | SGCrossover | SGCrossMutation | SinglePSO
[Name ^AlgoName][ENum ^ExperimentNum]
[ITER ^iterationnum][PSize ^popSize]
SEARCHSPACE[particle ^particle][Dim ^particle dimension]
PROBLEM[^FLPmodel |userdefined] [min ^ Min(distance|cost)]
```

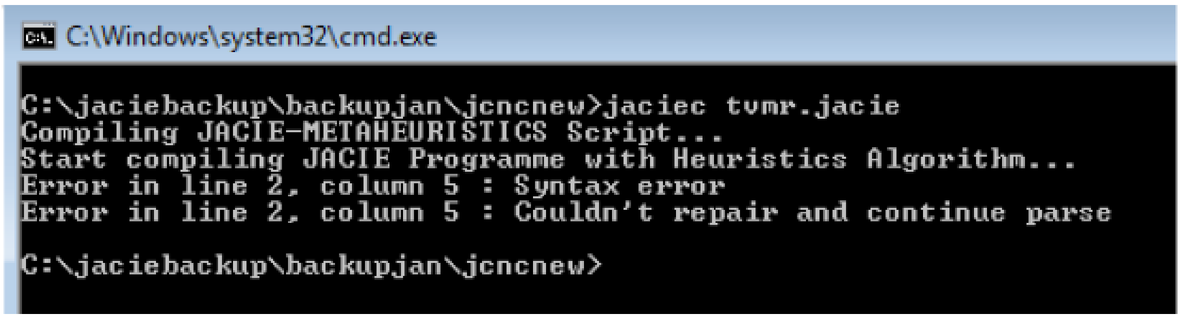
**Fig.1.** The compiler specifications for FLP in the general component

The general components are comprised of the predefined implementation framework [19] that has been used as a keyword for identifying the hybridization model. Developer can choose one from the three hybridization models and also allowed to choose the single PSO. The relevant parameters are governed by string variable *^AlgoName*, integer variable *^ExperimentNum*, integer variable *^IterationNum* and integer variable *^PopulationSize*.

Other important elements in the general specifications are search space and problem. *SEARCHSPACE* and *PROBLEM* are the two keywords used for defining the related parameters such as the solutions representation and the dimension of particles for the PSO in the search space. For defining the FLP model, the language keyword is *PROBLEM* and the parameters can be used to set the FLP model either QAP, MIP or GraphTheory followed with the problem objective governed by a Boolean variable *^Min (default min=true).* The variable *^ModelName* calls the related predefined FLP formulation available at the back-end software library. A more flexible option for problem definition is *userdefined* that permits user to insert new user-defined codes segment, providing programmers with the capability to incorporate new formulation of FLP.

### 3.1 Compilation error

The JACIE compiler have been provided with some mechanisms for checking any errors that the programmer may possibly codes. During its first phase of lexical analysis, the compiler checks all the tokens for the language. There are more than 300 tokens have been created in JACIE with more than 50 tokens are specifically used for the hybridization of PSO-GA. A programmer tends to make mistakes in writing tokens for example, missing comma, uncomplete parenthesis or wrong spelling for keywords. All results in an error detected by the lexical analyser forces the JACIE compiler to stop the execution. Once the scripts codes are starting to be compiled, JACIE processes the token specifications into a token table so that the compiler can check on all the tokens found in the program. Figure 2 is an example screen shot in compiling a program that has the error of wrong typing of keyword at line two. Once error has been found, JACIE stop the compilation process by displaying the line with error.



```
C:\Windows\system32\cmd.exe

C:\jaciebackup\backupjan\jcncnew>jaciec tvmr.jacie
Compiling JACIE-METAHEURISTICS Script...
Start compiling JACIE Programme with Heuristics Algorithm...
Error in line 2, column 5 : Syntax error
Error in line 2, column 5 : Couldn't repair and continue parse

C:\jaciebackup\backupjan\jcncnew>
```

**Fig.2.** Lexical errors

## 3. THE SCRIPTING LANGUAGE CODES

The paper describes the codes of the scripting language that used to solve FLP problems based on meta-heuristics PSO-GA. Our previous papers in [8] have reported about the functional evaluations of the scripting language that compared the optimization results of different PSO-GA hybrids and single PSO developed with the scripting languages and JAVA programming language. The optimization results for all algorithms generated from the scripting language codes are almost equivalent with the results generated from JAVA codes.

Figure 2 presents the complete scripting codes for solving FLP with QAP model and to use PSO-GA hybrid. The PSO-GA hybrid includes both crossover and mutation operations from the GA into the PSO. The scheme name is *SGCrossMutation* with *Time-vary Crossover Mutation (TCMR)*. Our previous paper in [20] explains time-vary parameterizations of the PSO-GA hybrid.

```
JACIE{
SGCrossMutation(Name TCMR, ENum 40,ITER 3000, PSize 40);
SEARCHSPACE(particle, dim 30);
PROBEM(FLP(QAP), min(distance));
UPDATE (inertia[const 0.4],c1[const 2],c2[const2],MaxP 10.0, MinP -5, MaxV
10, MinV
5);
Crossover (Crate[time-vary LD 0.6 1.0], C_operation[pbest],
Soperation[roulettwheel]);
Mutation(Mrate const 0.1, M_operation[Gaussian]);
}
```

**Fig.2.** The complete codes with the scripting language

The codes begin with JACIE keyword and the main body must enclose with open and close curly brackets. The scripting codes are much simpler than the JAVA codes in JSWARM software framework. Figure 3 presents the JAVA codes. The number of non-space characters written with the scripting language codes and with the JAVA are 320 and 2398 respectively. The scripting codes are very wordless but presentable enough to convey information about the algorithm flow and configurations. To compare the output, the total minimum distances of the FLP from the algorithm developed with the scripting language platform is 21499, which is within the range of results produced from the JAVA codes (21474). Therefore, the scripting language is workable to produce the desired results.

Based on the codes comparison, it can be concluded that the scripting language provides benefits to support easy programming environment for rapid prototyping and testing PSO-GA hybrid in solving the FLP. The proposed scripting language is easy to use because it is wordless with simple and straightforward statements to be used and comprehended by non-expert programming users. The statements and keywords are thoroughly designed to be closely alike to the relevant components of PSO, GA and FLP. Additionally, it has very small number of symbols, only with comma, semicolon, parenthesis and bracket. More important, the symbols and keywords are distinguishable each other in relation to different purposes. For example, *Mrate* and *Crate* are used for representing mutation rate and crossover rate respectively. Semicolon means to end a statement, curly bracket is used to begin and end the program, parameters of each statement are separated with comma within open and end parenthesis. Also, it uses common symbols for comment as used in the JAVA and C++ languages. To keep the codes easily understandable and to encourages algorithm designers to use abstractions for algorithmic variation, no conditional and repetition statements are included within the scripting code unless users must create new problem to be solved.

## 4.0 CONCLUSION

This paper provides general descriptions about the implementation of PSO-GA hybrids with scripting language programming platform in solving FLP. Compared to JAVA, the scripting language constructs has simple and concise statements, which facilitates fast prototyping for algorithm development and testing. However, the scripting language evaluation provided in this paper is not very informative to present the language readability or usability. A better approach in future work would have been to have human testing to test their comprehension with the terms used. A demographic analysis would also be possible to identify the users backgrounds, including variables such as the level of users understanding on FLP, PSO and GA as well as the type of reusable they use in the development and their programming skill.

```
import java.io.*;import net.sourceforge.jswarm_pso.*;
class TCMR{public static void main(String[] args) {
  double bestf=0.0; double stddev=0.0;double ch=0.0; Mutator mutator; Particle particles[];double
  evolspeed =0.0;double aggregation =0.0;
  ParticleUpdate particleUpdate; int numofexperiments = 30; Selection selection; onepointcrossover
  crossover;pbestcrossover crossover1;
  Particle particle1,particle2,Offstring;int conv=0;int avgconv=0;int totalconv=0;double prevfitness=
  0.0; double nextfitness=0.0; double[] chibestf = new double[numofexperiments];
for(int z=0;z<numofexperiments;z++){
            Swarm swarm = new Swarm(40, new MyParticle(), new MyFitnessFunction());
              swarm.setInertia(0.9); swarm.setGlobalIncrement(2.0);
              swarm.setParticleIncrement(2.0);swarm.setMaxPosition(10);         swarm.setMinPosition(-
              5);swarm.setMaxMinVelocity(10);
              mutator = new Gaussianmutator(swarm);int numberOfIterations =2000;
             double[] convergencearray = new double [numberOfIterations];
             double totalfit=0.0;double meanfitness=0.0;
             double s=0.1;double inertia=0.0; double h=0.1;
            int numofparticle = swarm.getNumberOfParticles();
for (int i = 1; i < numberOfIterations-1; i++) {
            Swarm.initialization();swarm.evaluate(); selection = new Selection (swarm);
             particle1 = selection.rouletewheel();
             particle2 = selection.rouletewheel(); double gbest = swarm.getBestFitness();
                                   for(int j=0;j<numofparticle;j++){
                                     TimeVary  TV = new TimeVary(gbest);
                                     double velocity[] = swarm.getParticle(j).getVelocity() ;
                                     double positionvalue[] = swarm.getParticle(j).getPosition();
                                    double fitness = swarm.getParticle(j).getFitness();
                                    double isavalue = adap3.ISA(positionvalue[1],fitness);
                                    inertia = 1 - ( 0.3 * (1/(1+ Math.exp(-(isavalue)))));
                                    double pbest1=particle1.getFitness();
                                    double pbest2=particle2.getFitness();
                                     crossover = new pbestcrossover(particle1, particle2);
                                    double crate=inertia;
                                    double cprob = crossover.probability(crate);
                                   Offstring = crossover.crossoverallposition(cprob);
                                   double mrate=inertia;
                                   double x = mutator.probability(swarm, mrate);
                                    mutator.mutatorallposition(swarm, x);
                                    mutator.mutatorallpositionoperation(swarm, x,inertia);
                                    swarm.update();}
       convergencearray[i-1] = new Double(swarm.getParticle(idxBest).toString());  }

             for(int cv=numberOfIterations-3; cv>=1;cv--)
               {prevfitness=convergencearray[cv];
                      if(cv+1 != 0)
                       nextfitness=convergencearray[cv-1];
                     else nextfitness=convergencearray[cv];
                   if (prevfitness != nextfitness)break;conv=cv;  }
                   System.out.println(swarm.toStringStats());
                   System.out.println("Convergence at iteration" + conv);
chibestf[z] = swarm.getParticle(swarm.getBestParticleIndex()).getBestFitness(); totalconv += conv;
}
```

**Fig.3.** The main JAVA codes within JSwarm Software Framework

## 5. REFERENCES

[1] A. Drira, H. Pierreval, and S. Hajri-Gabouj, "Facility layout problems: A survey," *Annu. Rev. Control*, vol. 31, no. 2, pp. 255–267, 2007.

[2] A. S. Abd Rahman, M. N. Zakaria, and S. Masrom, "Attack path selection optimization with adaptive genetic algorithms," *Adv. Sci. Lett.*, vol. 22, no. 10, pp. 2871–2875, 2016.

[3] S. Kulturel-Konak and A. Konak, "Linear programming based genetic algorithm for the unequal area facility layout problem," *Int. J. Prod. Res.*, vol. 51, no. 14, pp. 4302–4324, 2013.

[4] B. H. Ulutas and S. Kulturel-Konak, "An artificial immune system based algorithm to solve unequal area facility layout problem," *Expert Syst. Appl.*, vol. 39, no. 5, pp. 5384–5395, 2012.

[5] D. Datta, A. R. S. Amaral, and J. R. Figueira, "Single row facility layout problem using a permutation-based genetic algorithm," *Eur. J. Oper. Res.*, vol. 213, no. 2, pp. 388–394, 2011.

[6] J. F. Gonçalves and M. G. C. Resende, "A biased random-key genetic algorithm for the unequal area facility layout problem," *Eur. J. Oper. Res.*, vol. 246, no. 1, pp. 86–107, 2015.

[7] N. Yalaoui, L. Amodeo, H. Mahdi, and F. Yalaoui, "Hybrid method to solve a facility layout problem: Genetic algorithm-particle swarm optimization," *IFAC Proc. Vol.*, vol. 42, no. 4, pp. 1251–1255, 2009.

[8] S. Masrom, S. Z. Abidin, N. Omar, A. S. Rahman, and Z. I. Rizman, "Dynamic parameterizations of particle swarm optimization and genetic algorithm for facility layout problem," *ARPN J. Eng. Appl. Sci.*, vol. 12, no. 10, pp. 3195–3201, 2017.

[9] M. Fowler, *Domain Specific Languages*, 1st ed. Addison-Wesley Professional, 2010.

[10] S. Dower, "Disambiguating Evolutionary Algorithms:Composition and Communication with ESDL," University of Swinburne, 2011.

[11] A. Fink and S. Voß, " Hotframe: A Heuristic Optimization Framework ," in *Optimization Software Class Libraries*, vol. 18, Springer US, 2002, pp. 81–154.

[12] C. Voudouris, R. Dorne, D. Lesaint, and A. Liret, "iOpt: A Software Toolkit for Heuristic Search Methods ," in *Principles and Practice of Constraint Programming — CP 2001*, vol. 2239, T. Walsh, Ed. Springer-Verlag Berlin Heidelberg, 2001, pp. 716–729.

[13] S. Wagner and M. Affenzeller, "HeuristicLab: A Generic and Extensible Optimization Environment ," in *Adaptive and Natural Computing Algorithms*, B. Ribeiro, R. F. Albrecht, A. Dobnikar, D. W. Pearson, and N. C. Steele, Eds. Springer Vienna, 2005, pp. 538–541.

[14] S. Ventura, C. Romero, A. Zafra, J. A. Delgado, and C. Hervás, “JCLEC: a Java framework for evolutionary computation ,” in *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, vol. 12, Springer Berlin / Heidelberg, 2008.

[15] M. Lukasiewycz, M. Glaß, F. Reimann, and D.-I. Sabine Helwig, “The OPT4J Documentation,” 2009. .

[16] S. Masrom, S. Z. Z. Abidin, N. Omar, and Z. I. Rizman, “Software framework for optimization problems and meta-heuristics based on scripting language,” *J. Fundam. Appl. Sci.*, vol. 9, no. 5S, pp. 33–48, 2017.

[17] M. A. . Ismail, M. Chen, and P. . Grant, “JACIE—An Authoring Language For WWW-Based Collaborative Applications,” *Ann. Softw. Eng.*, vol. 12, no. 1, pp. 47–75, 2001.

[18] C. Pabl, “JSwarm-PSO,” 2006. [Online]. Available: http://jswarm-pso.sourceforge.net/.

[19] S. Masrom, A. S. A. Rahman, S. Z. Z. Abidin, N. Omar, and Z. I. Rizman, “The implementation frameworks of meta-heuristics hybridization with dynamic parameterization,” *J. Fundam. Appl. Sci.*, vol. 9, no. 6S, pp. 558–576, 2017.

[20] S. Masrom, S. Z. Z. Abidin, N. Omar, and K. Nasir, “Time-Varying Mutation in Particle Swarm Optimization,” in *Intelligent Information and Database Systems*, Springer-Verlag Berlin Heidelberg, 2013, pp. 31–40.