DIMMA:
A Design and Implementation Methodology for Metaheuristic Algorithms A Perspective from Software Development

Masoud Yaghini, Iran University of Science and Technology, Iran

Mohammad Rahim Akhavan Kazemzadeh, Iran University of Science and Technology, Iran

ABSTRACT

Metaheuristic algorithms will gain more and more popularity in the future as optimization problems are increasing in size and complexity. In order to record experiences and allow project to be replicated, a standard process as a methodology for designing and implementing metaheuristic algorithms is necessary. To the best of the authors' knowledge, no methodology has been proposed in literature for this purpose. This paper presents a Design and Implementation Methodology for Metaheuristic Algorithms, named DIMMA. The proposed methodology consists of three main phases and each phase has several steps in which activities that must be carried out are clearly defined in this paper. In addition, design and implementation of tabu search metaheuristic for travelling salesman problem is done as a case study to illustrate applicability of DIMMA.

Keywords: *Ant Colony Optimization, Design and Implementation Methodology, Evolutionary Algorithms, Genetic Algorithm, Metaheuristic Methodology, Operations Research, Tabu Search*

1. INTRODUCTION

Optimization problems, which occur in real world applications, are sometimes NP-hard. In the case of NP-hard problems, exact algorithms need, in the worst case, exponential time to find the optimum. Metaheuristics or *modern heuristics* deal with these problems by introducing systematic rules to escape from local optima. Metaheuristics are applicable to a wide range of optimization problems (Doreo et al., 2006; Morago, DePuy, & Whitehouse, 2006). Some popular population-based metaheuristic methods are genetic algorithm (Goldberg, 1989) and ant colony optimization (Dorigo & Stützle, 2004) in which collective intelligence play the important role (Wang, 2010). Tabu search (Glover & Laguna, 1997) and simulated annealing (Kirkpatrick, Gelatt, & Vecchi, 1983) are the two popular single-solution based metaheuristics that improve a single solution in an iterative algorithm.

With growing scale and complexity of optimization problems, metaheuristics will gain more and more popular. According to significant growth in using metaheuristics as optimization tools, there must be a standard methodology for design and implementing

DOI: 10.4018/jamc.2010040104

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

them. Such a methodology is used for recording experience and allows projects to be replicated. Moreover, this standard methodology can be a comfort factor for new adopters with little metaheuristic experience, and can show the guidelines to everyone who want to design and implement metaheuristics.

To the best of our knowledge, no methodology has been proposed in literature for design and implementation metaheuristic algorithms. There are many software frameworks in the literature for metaheuristics (Voss & Woodruff, 2002; Fink et al., 1999), in which framework means reusable programming codes and components for metaheuristics (Talbi, 2009). Hence, the meaning of frameworks in these references is different from our proposed methodology. Although there are several tutorials as lectures on how to design meheuristics (Thierens, 2008), they are sometimes for special metaheuristic and do not consider this process as a whole.

The proposed methodology in this paper, a *Design and Implementation Methodology for Metaheuristic Algorithms (DIMMA)*, shows guidelines to everyone who wants to design and implement a metaheuristic algorithm. Webster's collegiate dictionary defines methodology as "a body of methods, rules, and postulate employed by a discipline" or "the analysis of the principles or procedures of inquiry in a particular field" (Merriam-Webster, 1997).

DIMMA includes several phases, steps, disciplines, and principles to design and implement a specific metaheuristic for a given optimization problem. In other words, DIMMA refers to the methodology that is used to standardize process of design and implementing a metaheuristic algorithm. In Sections 2-5 weWe explain the architecture of DIMMA and its phases and steps, In Section 6 wefollowed by a description of each step of DIMMA using design and implementation of Tabu Search (TS) metaheuristic for Travelling Salesman Problem (TSP) as a case study.

2. ARCHITECTURE OF DIMMA

The architecture of DIMMA has been inspired from *Rational Unified Process* (RUP) which is a methodology for software engineering (Kroll & Krutchten, 2003). DIMMA has two dimensions including *dynamic* and *discipline* dimension (Figure 1). Dynamic dimension is the horizontal dimension, which includes phases of the methodology: *initiation*, *blueprint*, and *construction*. Discipline dimension is the vertical dimension that shows the disciplines, which logically group the steps, activities, and artifacts.

DIMMA has three sequential phases that each of them has several steps (Figure 2). In each step, we define several activities, which must be done to complete the steps. These phases are as follows: *initiation*, in which the problem in hand must be understood precisely, and the goal of designing metaheuristic must be clearly defined. The next phase is *blueprint,* the most important goals of this phase are selecting metaheuristic solution method, defining performance measures, and designing algorithm for our solution strategy. The last phase is *construction* in which implementing the designed algorithm, parameters tuning (parameter setting), analyzing its performance, and finally documentation of results must be done. In some steps, it is necessary to review pervious steps to justify and improve decisions and algorithm. For example, it is common for the algorithm to be modified after the performance evaluations. These backward movements are illustrated in Figure 2.

3. INITIATION PHASE

Step 1.1: State the Problem

Stating the problem is the step 1.1 in DIMMA that is helpful in narrowing the problem down and make it more manageable. To state the problem, one can write simple statement that includes one or more objectives, inputs, outputs,

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

Figure 1. Two dimensions of DIMMA and level of effort in each discipline during the phases

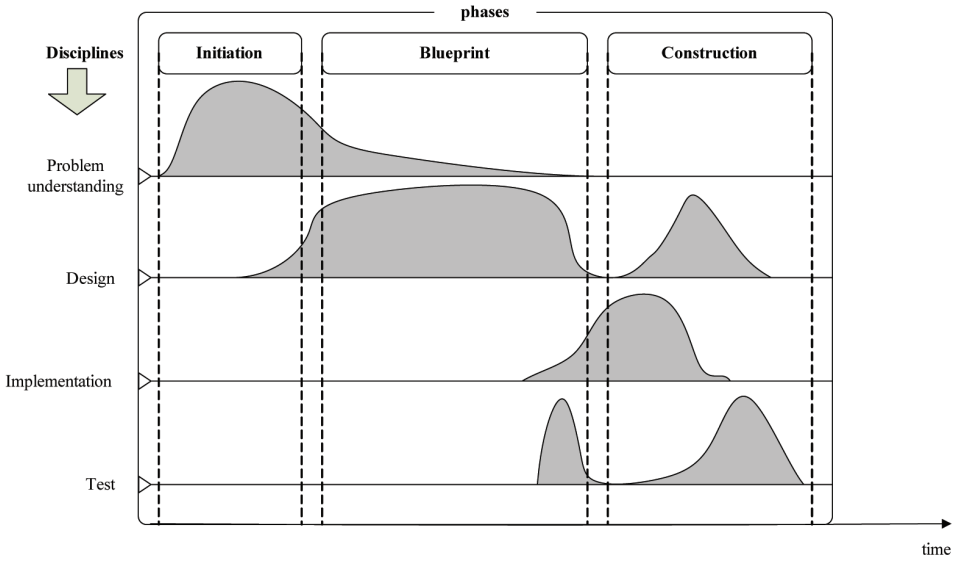
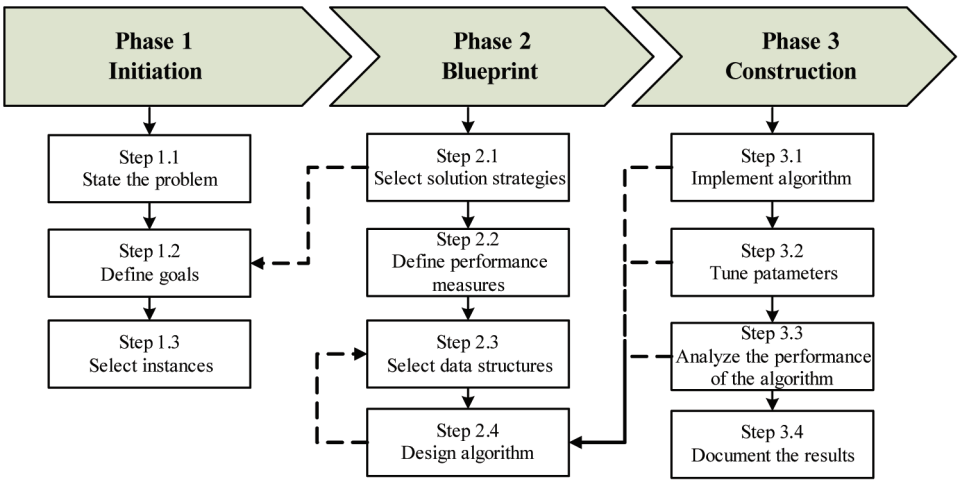


Figure 2. Steps in each phases of DIMMA



and assumptions of problem. In this step, a mathematical model can be provided for clarity. However, in some cases, it is difficult to formulate the problem with an unambiguous analytical mathematical notation.

The structure of problem such as multi-objective approaches, dynamic aspects, continuous or discrete modeling which is defined here can have significant effects on the next steps including defining goals, selecting solution strategy, and defining performance measures.

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

Step 1.2: Define Goals

In the step 1.2 of DIMMA, the goals of developing the metaheuristic must be defined clearly. All the experiments, performance analysis measures, and statistical analysis will depend on these goals. In addition, goal definition can be helpful in selecting solution strategy.

Some goals of designing the metaheuristic are: (1) reducing search time in comparison with exact methods or another metaheuristics, (2) improving quality of solutions, (3) robustness in terms of the instances, (4) solving large-scale problems, (5) easiness of implementation, (6) easiness to combine with other algorithms to improve performance, (7) flexibility to solve other problems or optimization models, and (8) providing a tight approximation to the problem. Selecting instances and solution method, and defining performance measures must be done according to selected goals. For example, if you want to reduce the search time, you must select time measurements in step 2.2, and you must select instances which can be comparable to another research works.

Step 1.3 Select Instances

In the step 1.3 of DIMMA, we must select input instances carefully to perform the evaluation of the algorithm. The chosen instances must be representative of the problem that one is trying to solve. The selected structure of input instances may influence the performance of metaheuristics significantly. To obtain interesting result and to allow the generalization of the conclusions, the selected instances must be diverse in terms of size of the instances, their complexity, and their structure (Alba & Lugue, 2005; Talbi, 2009; Silberholz & Golden, 2010). Keep in mind that according to No Free Lunch (NFL) theorem (Wolpert & Macready, 1997), no optimization algorithm is better than any other on all possible optimization problems; it means that, if algorithm *A* performs better than *B* for a given problem, there is no proof that *A* is always better than *B* and there is always another problem where *B* performs better than *A*. Therefore, the aim of designing an algorithm must be for a class of problems (Talbi, 2009).

Instances which are used for implementing and analyzing metaheuristics can be divided into *real-life* and *constructed* instances. Real-life instances are taken from real world applications. *Pure real-life* and *random real-life* instances are the two types of real life instances. Pure real-life instances are the practical instances of real world applications. If available, they are the good tools for evaluating the performance of a metaheuristic. Obtaining pure real-life instances is difficult, because those data aren't public, and collecting such data may be time consuming or expensive (Talbi, 2009; Silberholz & Golden, 2010). Random real-life instances are an alternative of real-life instances which use random real instances. In this type of instances, the structure of the real-life problem is preserved, but details are randomly changed to produce new instances (Alba & Lugue, 2005; Fischetti, Gonzalez, & Toth, 1997).

Standard instances and *pure random instances* are the two types of constructed instances. Standard instances are the instances that are widely used in experimentations, and because of that, became standard in literature. OR-Library (Beasley, 1990), MIPLIB (Achterberg, Koch, & Martin, 2003), and TSPLIB (Reinelt, 1991) are three examples of public libraries of standard instances which are available on Internet. By means of the standard instances we can compare our designed metaheuristic with another methods in literature (Alba & Lugue, 2005; Talbi, 2009). Finally, when none of the above instances are available, the remaining alternative is pure random instance generation. Although this type of instances can generate a wide range of instances with different size and complexity, they are often too far from real-life problems to reflect their structure and important characteristics and as a result, performance evaluation using only this type of instances might be controversial (Alba & Lugue, 2005; Gendreau, Laporte, & Semet, 1998).

After selecting instances, the selected instances must be divided into two subsets; one for tuning the metaheuristic parameters and

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

the second for evaluating the algorithm performance with tuned parameters. Tuning instances should be representative of the whole class of instances that the algorithm will eventually encounter. Therefore, since tuning instances are representative of another, the performance of algorithm on these two subsets of instances must be similar to each other. Also the instances which are used for tuning parameters must not be applied in performance evaluation because this may lead to controversial results that the parameters are just suitable for these problems (Biratteri, 2009).

In addition, after selecting instances, classifying them might be useful. Classify instances means categorizing them into classes according to some important factors such as size and complexity. Performance analysis of constructed metaheuristic algorithm must be done on each class of this problem instances (Silberholz & Golden, 2010). This kind of approach can be found in Hartmann and Kolisch (2000) in which three factors such as network complexity, resource factor, and resource strength are used to classify instances for project scheduling problem. Sometimes it is useful to use *fitness landscape analysis* for classifying the instance. This work may help to know the difficulties and structure of instances (see Stadler & Schnabl, 1992; Stadler, 1995).

4 BLUEPRINT PHASE

Step 2.1: Select Solution Strategy

In step 2.1 of DIMMA, after reviewing existing solution methods for the problem, the necessity of using metaheuristics must be specified. Indeed, according to the existing solution methods, it must be distinguished if applying metaheuristic for the problem is necessary or not. If a metaheuristic approach is selected as a solution method, one can go to the next step; otherwise we must stop in this point.

According to the situation, we can select one of the solution strategies such as exact algorithm, heuristic algorithm, metaheuristic algorithm, *hybrid method*, *parallel algorithm*, and *cooperative algorithm*.

For selecting metaheuristic or exact methods as a solution strategy, we must keep in mind four main factors. First, one is the *complexity* of a problem. Complexity can be seen as an indication of hardness of the problem. The second important factor is the *size* of input instances. In the case of small instances, the problem may be solved by an exact approach, even if the problem is NP-hard. The *structure* of the instances is another important factor. Some of the problems with specific structure by medium or even large dimensions may be solved optimally by exact algorithms. The last factor is the *required search time* to solve a given problem which must take into account. If we have real-time constraint (depend on problem, can be some seconds to some months), even if the complexity of problem is polynomial, the using of metaheuristic can be justified. If for a given problem, there is an exact or other state of the art algorithm in literature, selecting metaheuristic, as a solution strategy is not rational (Talbi, 2009).

In addition to above factors, ease of use of certain metaheuristic over certain problem can play an important role. For example, ACO is very intuitive for TSP, but may be difficult to adapt to continuous problems. Also, in some cases, the direct use of a generic metaheuristic would not lead to good results and sometimes there is a need to adapt it for the problem with the use of specialized heuristics.

Hybrid algorithms are one of the approaches to use metaheuristics as optimization tools. A hybrid algorithm is a combination of complete (exact) or approximate algorithms (or both) used to solve the problem in hand (El-Abd & Kamel, 2005). These methods are used when we want the specific advantages of different approaches.

The central goal of parallel computing is to speed up computation by dividing the workload among several processors. From the view point of algorithm design, *pure* parallel computing strategies exploit the partial order of algorithms (i.e., the sets of operations that may be executed

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

concurrently in time without modifying the solution method and the final solution obtained) and thus correspond to the *natural parallelism* present in the algorithm (Crainic & Toulouse, 2003). Cooperative algorithms are a category of parallel algorithms, in which several search algorithms are run in parallel in order to solve the optimization problem in hand. The search algorithms (run in parallel) may be different, that is why a cooperative search technique may be also viewed as a hybrid algorithm (El-Abd & Kamel, 2005).

After selecting solution strategy, we must specify algorithm components for our problem. This specification can be helpful in the step of selecting data structure and designing the algorithm.

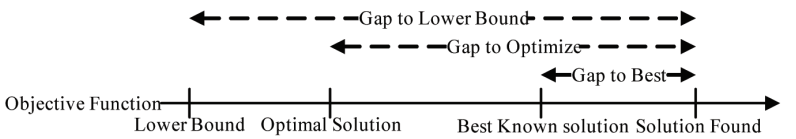
Step 2.2: Define Performance Measures

In this step, the performance measures are selected for the step of performance analysis. We must select the appropriate measures according to the selected goals of using metaheuristics. For exact solution methods, in which the global optimality is guaranteed, search time is the main indicator to evaluate the performances of the algorithms. But, in the case of metaheuristics which try to find near optimal solution in reasonable time, both solution quality and computational time are the two main indicators of performance (Alba & Lugue, 2005; Talbi, 2009). In metaheuristic search methods, the indicators to evaluate the effectiveness include *quality of solutions*, *computational effort*, and *robustness* (Barr et al., 1995).

Quality of solutions is based on measuring the distance to one of the following solutions (Figure 3): *global optimal solution*, *lower* (*upper*) *bound solution*, *best known solution*, and *requirements or actual implemented solution*. For constructed instances, the global optimal is known. In this case, the percentage of runs in which the result is equivalent to global optimal (success rate), can be the best indicator. However, usually this global optimal is not available, so the lower or upper bound can be used for minimization and maximization problems, respectively. Several relaxation methods such as Lagrangian relaxation can be used to find lower and upper bound solution. For some standard and popular problems, the best known solution is available in the literature which can be used as an indicator to evaluating the quality of solutions. For real world application, there might be predefined goal that can be as a quality measurement (Talbi, 2009).

Computational effort is the worst-case or average-case complexity and CPU time that can be used for theoretical and empirical analysis of algorithm efficiency. For theoretical analysis, one can use complexity theory to calculate the complexity of an algorithm. However, it is not always sufficient to analyze the computation effort theoretically. CPU time is a measurement to analyze the computational effort empirically. One of the disadvantages of CPU time is that it depends on computer characteristics and compiler in which the algorithm is compiled (Talbi, 2009; Silberholz & Golden, 2010). Therefore, many researchers use the number of objective function evaluations as a measurement of the computational effort, since it eliminates the effects of these characteristics (Talbi, 2009). However, number of evaluations of objective function is not also the best metric

Figure 3. Performance assessment of the quality of the solutions in a minimization problem. (Adapted from Talbi, 2009)



Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

for computation time, because it may not be the part that takes the most of computation time.

As we mentioned, another metric for performance evaluation is robustness. If the results of algorithm have no or small changes by deviating in the input instances, the algorithm is robust (Montgomery, 2005). Once the experimental results are obtained, standard deviation of solution quality must be considered as a measurement of robustness.

Performance measures for multiobjective optimization (measures for convergence of metaheuristics towards the Pareto frontier) can be found in Collette and Siarry (2005). Moreover, performance measures for parallel optimization can be found in Crainic & Tolouse (1998). In addition, performance metrics for comparison of metaheuristics, such as run time and quality of solution, is well discussed in Silberholz and Golden (2010). It must be stated here that statistical analysis which must be done to evaluate performance of algorithm is discussed in section 3.3.

Step 2.3: Select Data Structure

Before designing the algorithm, it is necessary to select a proper data structure in step 2.3 of DIMMA. Data structure is a scheme for organizing information in memory, for better algorithm efficiency (MacAllister, 2009; Puntambekar, 2009). Array, files, lists, trees, and tables are the important types of data structures. For instance, for TSP, as we will mention in section 6.2, for representing a solution, one can use an array with the length of number of cities.

Selecting the right data structure can make an enormous difference in the complexity of the resulting implementation. Pick the right data representation can make the programming easier. If we select wrong data structure for an algorithm, implementing it might be too time consuming (Cormen et al., 2002; Skiena & Revilla, 2003).

Step 2.4 Design Algorithm

In the step 2.4 of DIMMA, overall structure of algorithm must be specified. There are various ways for specifying an algorithm (Puntambekar, 2009). Using *natural language* is the simplest way for specifying an algorithm. In this case, we specify the algorithm simply with natural language. Although such a specification is very simple, it is not usually clear enough to implement. *Pseudocode* is another way of specifying an algorithm that is a combination of natural and programming language. Using pseudocode, one can specify algorithm more precisely than a natural language. Instead of two previous ways for specifying an algorithm, one can use *flowchart*. Flowchart is a graphical specification of an algorithm. After specifying the overall structure of algorithm, the correctness of it must be checked. This work can be done by using one small instance of valid input.

In addition to the above tools of description of an algorithm, the dynamic behaviours of metaheuristics can be described by means of some concepts from RUP, such as system sequence diagram (SSD) in the UML (Unified Modeling Language) (Siau & Halpin, 2001). A sequence diagram in UML is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart.

The algorithm must also be analyzed according to four factors including *complexity*, *space efficiency*, *simplicity* and *generality*. The number of steps of pseudocode needed to specify the algorithm can be the metrics for complexity. However, these metrics depend on programming language and style of pseudocode (Silberholz & Golden, 2010). Space efficiency is space or memory usage of an algorithm. Simplicity is an important factor, because the simpler the algorithm is, the easier it can be programmed and debugged. Finally, generality is applicability of an algorithm to a wide range of inputs (Puntambekar, 2009).

5. CONSTRUCTION PHASE

Step 3.1 Implement Algorithm

The implementation of an algorithm is done by suitable programming language in step 3.1 of

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

DIMMA. For example, if an algorithm consists of objects and related methods then it will be better to implement such algorithm using one of the object oriented programming language such as C++, C# or Java.

In this step, in order to speed out programming and reducing development costs, one can use many free software frameworks. Software frameworks are reusable programming codes and components for metaheuristics. One can use frameworks to implement a metaheuristic without in-depth knowledge of programming. Some of the software frameworks which have been proposed for single-objective problems are EasyLocal++ (Gaspero & Schaerf, 2001), Localizer++ (Michel & Van 2001), GAlib (Wall, 1996), MAFRA (Krasnogor & Smith, 2000), Hotframe (Fink, Voss, & Woodruff, 1999), iOpt (Voudouris et al., 2001), DREAM (Arenas et al. 2002), MALLBA (Alba et al., 2002), ECJ (Wilson, 2004), and CIlib (Cloete, Engelbrecht, & Pampar, 2008). EasyLocal++ and Localizer++ have been designed for local search algorithms, while GAlib is the framework for genetic algorithm. MAFRA is a framework for genetic local search algorithm (memtic algorithm). Hotframe is for evolutionary algorithms and metaheuristics which use single solution instead of population of solutions. Some of these frameworks including MALLBA, and ECJ are just for evolutionary algorithms. iOpt is a framework for genetic algorithm that can handle hybrid approaches. CIlib is a framework for swarm intelligence and evolutionary computing algorithms. Many frameworks, and reusable codes for Particle Swarm Intelligence (PSO) can be found in Particle Swarm Central[1]. There are also frameworks for several metaheuristics in MetaYourHeuristic (Yin, 2010) that can be used for new adopters in the field of metaheuristics.

In the case of multi-objective optimization some of software frameworks including PISA (Bleuler et al., 2003) and ParadiseEO[2] have been developed. However, ParadiseEO can also handle single-objective problems. There are also several genetic algorithm source codes for single and multi-objective problems in Kanpur Genetic Algorithms Laboratory website[3].

Step 3.2: Tune Parameters

Parameters are the configurable components of a metaheuristic algorithm. Metaheuristics are sensitive to the value of their parameters. Therefore, in step 3.2 of DIMMA, parameter tuning, also known as parameter setting, should be done. To do this, several number of numerical and/or categorical parameters, has to be tuned, and to do this scientific method and statistical analysis could and should be employed (Biratteri, 2009; Talbi, 2009; Barr et al., 1995).

In the most of research papers in the field of metaheuristics, parameters are tuned by hand in a trial-and-error procedure. This approach has several disadvantages such as: time-consuming, labor-intensive, and it need practitioner with special skills (Biratteri, 2009).

The proper values for the parameters depend on three main factors: the problem at hand, the instance of the problem, and the required search time. There are two different strategies for parameter tuning including *off-line* and *on-line* strategies.

In off-line tuning strategy, the parameters are set before the execution of metaheuristics and don't update during the execution. In this strategy one-by-one parameter tuning doesn't guarantee the optimality of parameters values, because there is no interaction between parameters. To overcome this problem, *Design of Experiments* (DoE) can be used (Fisher, 1935; Montgomery, 2005; Frigon, 1997; Antony, 2003; Box, 2005). In DoE approaches, there are some factors (parameters) that each of them has different levels (potential values of parameters). With a full factorial design, the best level of each factor can be obtained, but this work takes high computational time. However, there are several DoE approaches which reduce the number of experiments (Montgomery, 2005). DOE refers to the process of planning the experiment so that appropriate data that can be analyzed by statistical methods will be collected, resulting in valid and objective conclusions (Biratteri, 2009).

The three basic principles of DoE are replication, randomization, and blocking. Replication means a repetition of the basic experiment.

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

Replication has two important properties. First, it allows the experimenter to obtain an estimate of the experimental error. Second, if the sample mean is used to estimate the effect of a factor in the experiment, replication permits the experimenter to obtain a more precise estimate of this effect. Randomization means that both the allocation of the experimental material and the order in which the individual runs or trials of the experiment are determined randomly. Randomization usually makes this assumption valid. Blocking is a design technique used to improve the precision with which comparisons among the factors of interest are made. Often blocking is used to reduce or eliminate the variability transmitted from nuisance factors; that is, factors that may influence the experimental response but in which we are not directly interested (Montgomery, 2005). The important parameters in DoE approach are response variable, factor, level, treatment and effect. The response variable is the measured variable of interest. In the analysis of metaheuristics, the typically measures are the solution quality and computation time (Adenso-Díaz & laguna, 2006). A factor is an independent variable manipulated in an experiment because it is thought to affect one or more of the response variables. The various values at which the factor is set are known as its levels. In metaheuristic performance analysis, the factors include both the metaheuristic tuning parameters and the most important problem characteristics (Biratteri, 2009). A treatment is a specific combination of factor levels. The particular treatments will depend on the particular experiment design and on the ranges over which factors are varied. An effect is a change in the response variable due to a change in one or more factors (Ridge, 2007). Design of experiments is a tool that can be used to determine important parameters and interactions between them. Four-stages of DoE consist of screening and diagnosis of important factors, modeling, optimization and assessment. This methodology is called sequential experimentation, which is used to set the parameters in the DoE approach and has been used in this paper for the proposed algorithm in the case study section (Montgomery, 2005).

Another approach in off-line parameter tuning, is to formulate tuning of parameters of a metaheuristic as an optimization problem (Talbi, 2009). This problem can be seen as an independent problem that by optimizing it, the best values of parameters are obtained. In the case of off-line parameter tuning, Coy et al. (2000) proposed a procedure, based on statistical design of experiments and gradient descent that finds effective settings for parameters found in heuristics. Adenso-Diaz and Laguna (2006) proposed a procedure for parameter tuning by means of fractional design of experiments and local search. Hutter et al. (2009) proposed An Automatic Algorithm Configuration Framework (ParamILS) as a procedure for automatic parameter tuning. Another approach for off-line tuning parameters is machine learning that can be found in Biratteri (2009).

At different time of the search, different values for parameters are optimal. Because of this, an important drawback of off-line strategies is that the value or parameters are fixed and can't change during the search. Therefore, online approaches that update parameters dynamically (a random or deterministic update of the parameter values without take into account the search process) or adaptively (changes the values according to the search progress using the memory of the search) during the search must be designed. In adaptive approaches, the parameters are encoded into the representation of solutions and as a result by changing solution the value of the parameters are also changed.

Resent works on tuning parameters for metaheuristic algorithms can be found in Birattari et al. (2002), Bartz-Beielstein (2006), Fukunaga (2008), and Hutter, Hoos, and Stützle (2007).

Step 3.3: Analyze the Performance of Algorithm

In step 3.3 of DIMMA, we must obtain the experimental results for different indicators to analyze performance of metaheuristic algo-

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

rithms with statistical tests, such as *t-test*, and ANOVA models for a comparison of more than two algorithms (Cohen, 1995). To use these tests, we must obtain some aggregation number that summarizes the average and deviation tendencies. These statistical tests are used to determine whether obtained conclusion is due to a sampling error or not. The selection of a given statistical hypothesis-testing tool is performed according to the characteristics of the data (Montgomery, 2005). Generally, it is not sufficient to analyze an algorithm based only on theoretical approach, so that empirical performance analysis is a necessary task to perform and must be done on a fair basis (Bartz-Beielstein, 2006; Rardin & Uzsoy, 2001; Dean & Voss, 1999). Many trials (at least 10, more than 100 if possible) must be carried out to derive significant statistical results. From this set of trials, many measures may be computed (Talbi, 2009) such as mean, median, minimum, maximum, standard deviation, the success rate that the reference solution (e.g., global optimum, best known, given goal) has been attained, and so on.

Step 3.4: Document the Results

In this step of DIMMA, the documentation must be done. The interpretation of the results must be explicit and driven using the defined goals and considered performance measures. Generally, graphical tools are more understandable than presenting the large amount of data results using tables. Some popular graphical tools include interaction plots, scatter plots, and box plots (Talbi, 2009).

Interaction plots represent the interaction between different factors and their effect on the obtained response (performance measure). Scatter plot is a tool to illustrate the compromise between various performance indicators. For instance, the plots display quality of solutions versus time, or time versus robustness, or robustness versus quality. Box plot illustrate the distribution of the results through their five-number summaries: the smallest value, lower quartile (Q1), median (Q2), upper quartile (Q3), and largest value. Box plot is useful in detecting outliers and indicating the dispersion of the output data without any assumptions on the statistical distribution of the data. In addition, scatter plot is useful to illustrate compromise between different performance indicators (Tufte, 2001).

6. A CASE STUDY

In this section, we design and implement tabu search metaheuristic for travelling salesman problem as a case study to illustrate applicability DIMMA. The reason of choosing TS and TSP is that they are well known in the literature.

Initiation Phase

In Step 1.1, we state the TSP problem. Given a list of *m* cities and their pairwise distances (or costs), the problem is to find a minimum tour that visits each city exactly once. In the other word, we want to find a minimum Hamiltonian tour between a set of cities. According to the direction of arcs between cities, the TSP problem is divided into two categories: *symmetric* and *asymmetric*. In symmetric TSP, the arcs are undirected; while in the asymmetric TSP, the arcs are directed and the costs are depend on the directions. In this section, the symmetric TSP is chosen as a case study.

For clarity, here a mathematical formulation is presented for the symmetric TSP. Let $x_j \in \{0,1\}$ *be the de*cision variable where j runs through all arcs A of the undirected graph and c_j is the cost of traveling that arc. To find a tour in this graph, one must select a subset of arcs such that every city is contained in exactly two of the arcs selected. The problem can therefore be formulated as:

$$\min \frac{1}{2} \sum_{j=1}^{m} \sum_{k \in J(j)} c_k x_k \quad (1)$$

Subject to:

$$\sum_{k \in J(j)} x_k = 2 \quad \forall j = 1, \ldots, m \quad (2)$$

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

$$\sum_{j\in A(K)} x_j \le |K| - 1 \quad \forall K \subset \{1,\ldots,m\} \qquad (3)$$

$$x_j = 0 \text{ or } 1 \quad \forall j \in A \qquad (4)$$

where $J(j)$ is the set of all undirected arcs connected to city j and $A(K)$ is the subset of all undirected arcs connecting the cities in any proper, nonempty subset K of all cities. The objective function (1) is to minimize the tour length. Constraint (2) ensures that every city is contained in exactly two of the selected arc. Indeed, this constraint ensures that the selected arcs construct a tour. Constraint (3) is a sub-tour elimination constraint that prevents construction of sub-tours.

In the next step, the goal of solving TSP should be defined. Finding a minimum tour in acceptable time is the goal of this case study.

In step 1.3, the TSP instances are selected. We choose instances from TSPLIB (Reinelt, 1991) that is one of the popular sources for TSP instances (Table 1). Selected instances must have enough diversity in terms of size. Therefore, we select 12 instances from TSPLIB with different sizes.

Blueprint Phase

In the first step of blueprint phase, solution strategy should be selected. Because the TSP is an NP-hard problem (Garey & Johnson, 1979), so that we use approximate a metaheuristic algorithm to solve it. In this example, to illustrate DIMMA, we use tabu search (Glover & Laguna, 1997; Gendreau, 2003) as a solution method for TSP. It should be noted that, according to the structure of TSP, some other metaheuristic could be used to solve it.

The main components of TS are *representation of solution*, *neighborhood structure* (*move*), *tabu list*, *tabu tenure*, *aspiration criteria*, *termination condition*, *intensification*, and *diversification strategies*.

In step 2.2 the performance measures for solving TSP with TS is defined. The defined goal is to find a good solution in a reasonable time. Therefore, as a case study, performance measures which are used for this example are solution quality and CPU time. The global optima are known for the selected instances (Table 1). Therefore, for solution quality measure we use error rate from global optima.

In step 2.3, data structure is selected. To solve TSP with tabu search, one can use an integer array for representation of individuals, which shows the permutation of the cities (Figure 4). To store this array and its length, *currentTour*[] and *currentTourLength* are considered in data structure, respectively.

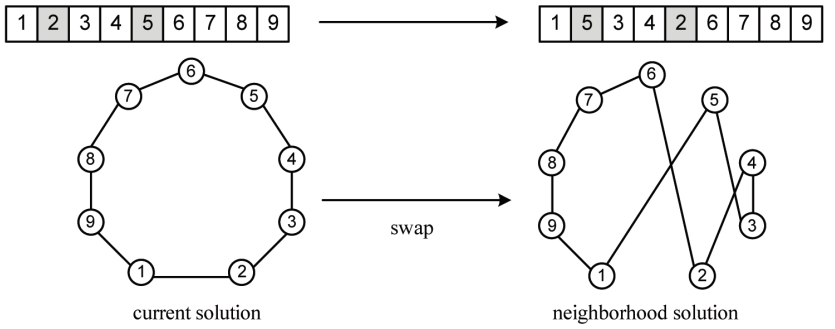
To construct initial solution we use nearest neighborhood heuristic method (Dorigo & Stützle, 2004). Therefore, we need an array and a variable to store them. Integer array *nearestNeighborArray*[] and integer variable *nearestNeighborTourLength* are used for this purpose.

Table 1. Problem instances for TSP from TSPLIB

#	Problem name	Number of cities	Global optima	#	Problem name	Number of cities	Global optima
1	ulysses16	16	6859	7	gr202	202	40160
2	ulysses22	22	7013	8	a280	280	2579
3	eil51	51	426	9	pcb442	442	50778
4	berlin52	52	7542	10	gr666	666	294358
5	kroA100	100	21282	11	pr1002	1002	259045
6	rd100	100	7910	12	u1060	1060	224094

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

Figure 4. Neighborhood structure for TS to solve TSP



For calculating tour length, we need distances between cities, so that, two dimensional integer array *distancesMatrix*[][] is used to store distances.

A neighborhood solution can be obtained by swapping two positions of cities in representation array (Figure 4). Therefore, neighborhood of a solution is all of the solutions that can be obtained by one swap.

The characteristics that must be saved in tabu list are the pair of cities that recently swapped. In this problem, the tabu list is a two dimensional array, *tabuList*[][]. In this array, the tabu tenure of swapping two cities i and j are stored in i^{th} raw and j^{th} column (Figure 5a). For example, the value in 6th raw and 9th column in Figure 5a shows the tabu tenure for swapping 6th and 9th cities. It means cities 6 and 9 cannot swap for 2 next iterations.

In this example, aspiration criteria is that if the result of a tabu move is better than that of the current best-known solution, then this move is allowed. For diversification we use *frequency based memory* (Glover, 1997), in this memory the information of swapping frequencies are stored (Figure 5b). For example, the value in 6th raw and 9th column in Figure 5b shows the frequency of swapping cities 6 and 9 in 70 iteration. According to these frequencies, one can assign penalty to swap pair cities. The more frequency is the more penalties are assigned. Therefore, the search can do swaps that are rarely done. Finally, for termination condition a number of iteration is chosen. To do this, integer variable *notImproveBestSoFar* is used in data structure. In step 2.4 of DIMMA TS algorithm is designed. We use pseudocode to specify TS (Figure 6).

Figure 5. (a) Tabu list for TS to solve TSP, (b) frequency based memory for diversification

(b)

	1	2	3	4	5	6	7	8	9
1		0	2	3	3	0	1	1	4
2			2	1	3	1	1	0	1
3				2	3	3	4	0	0
4					1	1	2	1	4
5						4	2	1	3
6							3	1	2
7								6	0
8									6
9									

(a)

	1	2	3	4	5	6	7	8	9
1				1					
2						5			
3							4		
4					3				
5									
6									2
7									
8									
9									

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

Construction Phase

In step 3.1 of DIMMA, TS algorithm is implemented. For implementing the algorithm, we select Java as a programming language. For coding algorithm we used NetBeens integrated development environment. We construct 2 classes (TSPByTS and Execution) and 20 methods for our algorithm.

In step 3.2, parameters of the TS algorithm are tuned. In this example, we use a DoE approach and Design-Expert statistical software (Vaughn et al., 2000) to obtain optimal values for parameters.

To tune parameters for the TS algorithm we classified instances into two classes, according to the number of cities. The instances that have less than or equal to 100 cities are categorized in class 1, and the other instances are classified into class 2. For the first class, instances kroA100 and eil51 and for the second-class a280 and pr1002 are chosen as representatives for parameter tuning. In the TS algorithm, solution quality and CPU time are considered as the response variables. Factors, their levels, and the final obtained values for classes 1 and 2 instances are summarized in Table 2 and Table 3. Each block is considered with 16 treatments and main effects.

Figure 6. The TS pseudocode for TSP

```
Start
/* generate initial feasible solution by nearest neighbor algorithm*/
Set currentTour[]= generate InitialSolution();
Set currentTourLength = computeTourLength(currentTour[]);
Set bestTourLength = currentTourLength;
Initialize tabuList and frequency based memory;
While (termination condition not met)
    Set currentTour[] = move (currentTour[]);
    Set currentTourLength = computeTourLength(currentTour[]);
    If (currentTour[] < bestTourLength), then bestTourLength = currentTourLength;
        /* save the best so far solution */
    If (iteration criterion met), then do diversification
    update tabuList and frequency based memory;
End While;
Output bestTourLength;
End.
```

Table 2. Factors, their levels, and the final obtained values for instances of class 1

Parameter	Lowest value	Highest value	Final Value
Tabu tenure	5	15	10
Termination Condition*	5000	40000	30000
Diversification condition*	1000	10000	3000

* The number of iteration that the best so far solution is not improved

Table 3. Initial, ranges, and final values for instances of class 2

Parameter	Initial value	Range of change	Final Value
Tabu tenure	5	15	14
Termination Condition*	1000	5000	3000
Diversification condition*	1000	5000	2000

* The number of iteration that the best so far solution is not improved

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

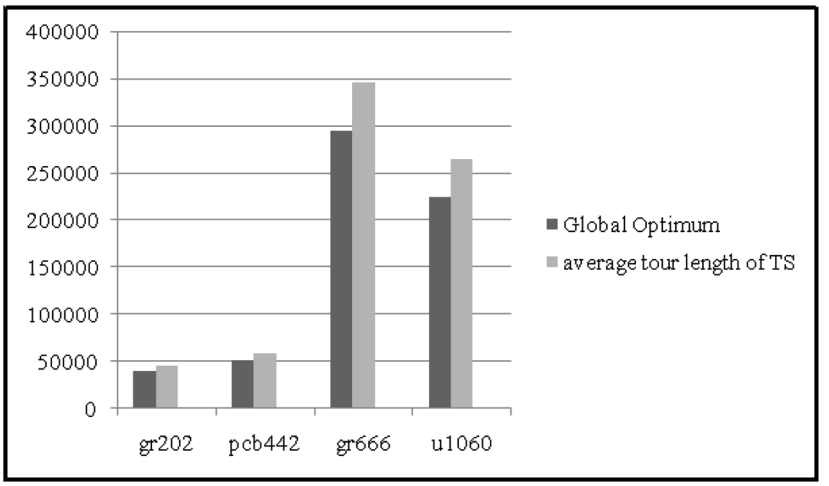
Table 4. Result of instance of class 1

Test problem	Global optimal	Average tour length	Average relative gap	Average CPU time (ms)	Best solution	Relative gap
ulysses16	6859	6859.00	0.0000	191.5	6859.0	0.0000
ulysses22	7013	7013.00	0.0000	449.6	7013.0	0.0000
berlin52	7542	7744.05	0.0267	5039.2	7544.3	0.0003
kroA100	21282	22170.47	0.0417	40847.1	22104.1	0.0492
rd100	7910	8232.51	0.0408	9111.1	8217.6	0.0389
Average			0.0218			0.0177

Table 5. Result of instance of class 2

Test problem	Global optimal	Average tour length	Average relative gap	Average CPU time (ms)	Best solution	Relative gap
gr202	40160	43304.29	0.0783	9419.4	41787.7	0.0405
pcb442	50778	58197.84	0.1461	60771.5	57386.2	0.1301
gr666	294358	338786.10	0.1509	99886.7	336833.6	0.1443
u1060	224094	256958.92	0.1467	159526.2	252441.6	0.1265
Average			0.1305			0.1104

Figure 8. global optimum and obtained solutions in each instances of class 2



Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

In step 3.3, the performance of TS algorithm is analyzed. We run the algorithm ten times with tuned parameters. The results are summarized in Table 4 and Table 5. The relative gap is calculated as follows:

$$relative\ gap = \frac{obtained\ solution - global\ optima}{global\ optima}$$

The step 3.4 of DIMMA is documentation that can be done by means of graphical tools. Figure 7 shows a sample to illustrate the performance of TS for TSP during the time for problem eil51. Figure 8 also compares the obtained solutions with global optimal in each instance. The figure illustrates that the algorithm can reach to near optimal.

CONCLUSION

According to significant growth in using metaheuristics as optimization tools, there must be a standard methodology for implementing them. Such a methodology is used for recording experience and allows projects to be replicated. Moreover, this standard process can be a comfort factor for new adopters with little metaheuristic background.

We have proposed the *DIMMA* as a methodology to design and implement metaheuristic algorithms. The proposed methodology includes series of phases, activities, disciplines, and principles to design and implement a specific metaheuristic for a given optimization problem. In the other word, the DIMMA refers to the methodology that is used to standardize process of design and implementing a metaheuristic.

We hope the proposed methodological approach to design and implementation of metaheuristics will draw more researchers to standardization of developing metaheuristics.

REFERENCES

Achterberg, T., Koch, T., & Martin, A. (2003). *The mixed integer programming library: Miplib*. Retrieved from http://miplib.zib.de

Adenso-Díaz, B., & laguna, M. (2006). Fine-tuning of algorithms using fractional experimental design and local search. *Operations Research*, *54*(1), 99–114. doi:10.1287/opre.1050.0243

Alba, E., Almeida, F., Blesa, M., Cotta, C., D'ıaz, M., Dorta, I., et al. Le'on, C., Moreno, L., Petit, J., Roda, J., Rojas, A., & Xhafa, F. (2002). MALLBA: A library of skeletons for combinatorial optimization. In B. Monien & R. Feldman (Eds.), *Euro-Par 2002 Parallel Processing Conference* (LNCS 2400, pp. 927-932). Berlin: Springer.

Alba, E., & Lugue, G. (2005). Measuring the performance of parallel metaheuristics . In Alba, E. (Ed.), *Parallel metaheuristics: A new class of algorithm* (pp. 43–60). New York: John Wiley & Sons.

Antony, J. (2003). *Design of experiments for engineers and scientists*. Barlington, UK: Butterworth-Heinemann.

Arenas, M. G., Collet, P., Eiben, A. E., Jelasity, M., Merelo, J. J., Paechter, B., et al. (2002). A framework for distributed evolutionary algorithms. In *Parallel Problem Solving from Nature Conference (PPSN VII)* (LNCS 2439, pp. 665-675). Berlin: Springer.

Barr, R. S., Golden, B. L., Kelly, J. P., Resende, M. G. C., & Stewart, W. R. (1995). Designing and reporting computational experiments with heuristic methods. *Journal of Heuristics*, *1*(1), 9–32. doi:10.1007/BF02430363

Bartz-Beielstein, T. (2006). *Experimental research in evolutionary computation*. New York: Springer.

Beasley, J. E. (1990). OR-Library: distributing test problems by electronic mail. *The Journal of the Operational Research Society*, *41*(11), 1069–1072.

Birattari, M., Stuetzle, T., Paquete, L., & Varrentrapp, K. (2002). A racing algorithm for configuring metaheuristics. In W. B. Langdon et al. (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002)* (pp. 11-18). San Francisco: Morgan Kaufmann Publishers.

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

Biratteri, M. (2009). *Tuning Metaheuristics: A machine learning perspective*. Heidelberg, Germany: Springer.

Bleuler, S., Laumanns, M., Thiele, L., & Zitzler, E. (2003). PISA: A platform and programming language independent interface for search algorithms. In *Proceedings of the Conference on Evolutionary Multi-Criterion optimization (EMO'03)*, Faro, Portugal (pp. 494-508).

Box, G., Hunter, J. S., & Hunter, W. G. (2005). *Statistics for experimenters: design, innovation, and discovery*. New York: Wiley.

Cloete, T., Engelbrecht, A. P., & Pampar, G. (2008). *CIlib: A collaborative framework for computational intelligence algorithms – part I*. Retrieved from http://www.cilib.net/

Cohen, P. R. (1995). *Empirical methods for artificial intelligence*. Cambridge, UK: MIT Press.

Collette, Y., & Siarry, P. (2005). Three new metrics to measure the convergence of metaheuristics towards the Pareto frontier and the aesthetic of a set of solutions in biobjective optimization. *Computers & Operations Research*, *32*, 773–792. doi:10.1016/j.cor.2003.08.017

Cormen, T. H., Leiserson, C. E., Rivest, R. L., & Stein, C. (2002). *Introduction to algorithms*. London: MIT Press.

Coy, S., Golden, B. L., Runger, G. C., & Wasil, E. A. (2000). Using experimental design to find effective parameter settings for heuristics. *Journal of Heuristics*, *7*, 77–97. doi:10.1023/A:1026569813391

Crainic, T. G., & Tolouse, M. (1998). Parallel metaheuristic . In Crainic, T. G., & Laporte, G. (Eds.), *Fleet management And logistic* (pp. 205–235). Norwell, MA: Kluwer Academic publishers.

Crainic, T. G., & Tolouse, M. (2003). Parallel Strategies FOR Meta-heuristics . In Glover, F., & Kochenberger, G. (Eds.), *Handbook of metaheuristics* (pp. 475–514). Norwell, MA: Kluwer Academic Publishers.

Doreo, J., Siarry, E., Petrowski, A., & Taillard, E. (2006). *Metaheuristics for hard optimization*. Heidelberg, Germany: Springer.

Dorigo, M., & Stützle, T. (2004). *Ant colony optimization*. Cambridge, UK: MIT Press.

El-Abd, M., & Kamel, M. (2005). A taxonomy of cooprative search algorithms . In Blesa, M. J., Blume, C., Roli, A., & Samples, M. (Eds.), *Hybrid metaheuristic* (pp. 32–42). Heidelberg, Germany: Springer. doi:10.1007/11546245_4

Fink, A., Voss, S., & Woodruff, D. L. (1999). Building reusable software components for heuristic search . In Kall, P., & Luthi, H. J. (Eds.), *Operations Research Proceedings* (pp. 210–219). Heidelberg, Germany: Springer.

Fischetti, M., Salazar Gonzalez, J. J., & Toth, P. (1997). A branch-and-cut algorithm for the symmetric generalized traveling salesman problem. *Operations Research*, *45*(3), 378–394. doi:10.1287/opre.45.3.378

Fisher, W. (1935). *The Design of Experiments*. Edinburgh, UK: Oliver and Boyd.

Frigon, N. L., & Mathews, D. (1997). *Practical guide to experimental design*. New York: Wiley.

Fukunaga, A. (2008). Automated discovery of local search heuristics for satisfiability testing. *Evolutionary Computation*, *16*(1), 31–61. doi:10.1162/evco.2008.16.1.31

Garey, M. R., & Johnson, D. S. (1979). *Computers and intractability*. San Francisco: Freeman and Co.

Gaspero, L. Di, & Schaerf, A. (2001). EasyLocal++: An object-oriented framework for the design of local search algorithms and metaheuristics. In *Proceedings of the MIC'2001 4th Metaheuristics International Conference*, Porto, Portugal (pp. 287-292).

Gendreau, M. (2003). An introduction to tabu search . In Glover, F., & Kochenberger, G. A. (Eds.), *Handbook of metaheuristics* (pp. 37–54). Norwell, MA: Kluwer Academic publishers.

Gendreau, M., Laporte, G., & Semet, F. (1998). A tabu search heuristic for the undirected selective travelling salesman problem. *European Journal of Operational Research*, *106*(2-3), 539–545. doi:10.1016/S0377-2217(97)00289-0

Glover, F., & Laguna, M. (1997). *Tabu search*. Dordrecht, The Netherlands: Kluwer Academic Publishers.

Goldberg, D. E. (1989). *Genetic algorithms in search, optimization and machine learning*. Reading, MA: Addison Wesley.

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

Hartmann, S., & Kolisch, R. (2000). Experimental evaluation of state-of-the-art heuristics for the resource-constrained project scheduling problem. *European Journal of Operational Research*, *127*(2), 394–407. doi:10.1016/S0377-2217(99)00485-3

Hutter, F., Hoos, H. H., Leyton-Brown, K., & Stützle, T. (2009). ParamILS: An automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, *36*, 267–306.

Hutter, F., Hoos, H. H., & Stützle, T. (2007). Automatic algorithm configuration based on local search. *AAAI*, 1152-1157.

Kirkpatrick, S., Gelatt, C., & Vecchi, M. (1983). Optimization by simulated annealing. *Science*, *220*, 671–680. doi:10.1126/science.220.4598.671

Krasnogor, N., & Smith, J. (2000). MAFRA: A Java memetic algorithms framework . In Freitas, A. A., Hart, W., Krasnogor, N., & Smith, J. (Eds.), *Data Mining with Evolutionary Algorithms* (pp. 125–131). Las Vega, NV.

Kroll, P., & Krutchten, P. (2003). *The Rational unified process Made Easy*. Reading, MA: Addison-Wesley.

MacAllister, W. (2009). *Data Structures and algorithms using java*. New York: Jones & Bartlett publishers.

Merriam-Webster. (1997). *Merriam-Websters's Collegiate Dictionary*. Merriam-Websters.

Michel, L., & Van, P. (2001). *Hentenryck. Localizer++: An open library for local search* (Tech. Rep. No. CS-01-02). Providence, RI: Department of Computer Science, Brown University.

Montgomery, D. (2005). *Design and analysis of experiments*. New York: Wiley.

Morago, R. J., DePuy, G. W., & Whitehouse, G. E. (2006). A solution methodology for optimization problems. In A. B. Badiru (Ed.), *Metaheuristics* (pp. 1-10, 13). New York: Taylor & Francis Group.

Puntambekar, A. A. (2009). *Analysis of algorithm and design*. New York: technical publications pune.

Rardin, R. L., & Uzsoy, R. (2001). Experimental evaluation of heuristic optimization. *Journal of Heuristics*, *7*(3), 261–304. doi:10.1023/A:1011319115230

Reinelt, G. (1991). TSPLIB: a traveling salesman problem library. *ORSA Journal on Computing, 3*, 376-384. Retrieved from http://softl ib.rice.edu/softlib/tsplib/

Ridge, E. (2007). *Design of experiments for the tuning of optimization algorithms*. Unpublished doctoral dissertation, Department of Computer Science, University of York, UK.

Siau, K., & Halpin, T. (2001). *Unified Modeling Language: system analysis, design and development issues*. Hershey, PA: IGI Global.

Silberholz, J., & Golden, B. (2010). Comparison of metaheuristics . In Gendreau, M., & Potvin, J.-V. (Eds.), *Handbook of metaheuristics*. Heidelberg, Germany: Springer. doi:10.1007/978-1-4419-1665-5_21

Skiena, S. S., & Revilla, M. A. (2003). *Programming challenges: The programming contest training manual*. New York: Springer.

Stadler, P., & Schnabl, W. (1992). The landscape of the traveling salesman problem. *Physics Letters. [Part A]*, *161*, 337–344. doi:10.1016/0375-9601(92)90557-3

Stadler, P. F. (1995). Towards a theory of landscapes. In R. Lop'ez-Pe˜na, R. Capovilla, R. Garc'ıa- Pelayo, H. Waelbroeck, & F. Zertuche (Eds.), *Complex Systems and Binary Networks* (Vol. 461, pp. 77-163). Berlin: Springer.

Talbi, E. (2006). *Parallel combinatorial optimization*. Hoboken, NJ: John Wily & Sons. doi:10.1002/0470053925

Talbi, E. (2009). *Metaheuristics: from design to implementation*. Hoboken, NJ: John Wiley & sons.

Thierens, D. (2008). From Multi-start Local Search to Genetic Local Search: a Practitioner's Guide. In *Proceedings of the 2nd International Conference on Metaheuristics and Nature Inspired Computing (META'08)*. Tunisia: Hammamet.

Tufte, E. R. (2001). *The Visual Display of Quantitative Information* (2nd ed.). Cheshire, CN: Graphics Press.

Vaughn, N., Polnaszek, C., Smith, B., & Helseth, T. (2000). *Design-Expert 6 User's Guide*. Stat-Ease Inc.

Voss, S., & Woodruff, D. L. (2002). *Optimization software class libraries*. Norwell, MA: Kluwer.

Voudouris, C., Dorne, R., Lesaint, D., & Liret, A. (2001). iOpt: A software toolkit for heuristic search methods. In *Proceedings of the International Conference on Principles and Practice of Constraint Programming* (LNCS 2239, pp. 716-729). Berlin: Springer.

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.

Wall, M. (1996). *GAlib: A C++ library of genetic algorithm components (Tech. Rep.)*. Mechanical Engineering Department, Massachusetts Institute of Technology.

Wang, Y. (2010). A Sociopsychological Perspective on Collective Intelligence in Metaheuristic Computing. *International Journal of Applied Metaheuristic Computing*, *1*(1), 110–128.

Wilson, G. C., McIntyre, A., & Heywood, M. I. (2004). Resource review: Three open source systems for evolving programs—Lilgp, ECJ and grammatical evolution. *Genetic Programming and Evolvable Machines*, *5*(19), 103–105. doi:10.1023/B:GENP.0000017053.10351.dc

Wolpert, D. W., & Macready, W. G. (1997). No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, *1*(1), 67–82. doi:10.1109/4235.585893

Yin, P. Y. (2010). *MetaYourHeuristic V. 1.3, Intelligence Computing Laboratory, National Chi Nan University, Taiwan*. Retrieved from http://intelligence.im.ncnu.edu.tw

ENDNOTES

1 http://www.particleswarm.info/Programs.html
2 http://paradiseo.gforge.inria.fr
3 http://www.iitk.ac.in/kangal/codes.shtml

Masoud Yaghini is Assistant Professor of Department of Rail Transportation Engineering, School of Railway Engineering, Iran University of Science and Technology. His research interests include data mining, optimization, metaheuristic algorithms, and application of data mining and optimization techniques in rail transportation planning. He published several books and papers in the field of data mining, metaheuristics, and rail transportation planning. He is teaching data mining, advanced operations research, and metaheuristic algorithms postgraduate courses.

Mohammad Rahim Akhavan Kazemzadeh is a MSc. Student in Rail Transportation Engineering, School of Railway Engineering, Iran University of Science and Technology. His research interests are metaheuristics optimization methods, parameter tuning of metaheuristics, multicommodity network design problems, and optimization in rail transportation problems. He has one underpublishing book in the field of metaheuristics and some published papers in the field of network design and metaheuristics.

Copyright © 2010, IGI Global. Copying or distributing in print or electronic forms without written permission of IGI Global is prohibited.