An Agent-based Approach for Dynamic Combination and Adaptation of Metaheuristics

Tjorben Bogon, Andreas D. Lattner, Yann Lorion, Ingo J. Timm

Professur für Wirtschaftsinformatik und Simulation,
Goethe Universität Frankfurt am Main

1 Introduction

Many organizations have to face optimization problems in their day-to-day business. Popular examples are scheduling problems where jobs have to be assigned to machines or employees to shifts or vehicle routing problems (VRPs) (Lenstra and Kan 2006) where it has to be decided how and in which order goods are delivered to (and/or picked up from) different stations, taking into account a number of constraints. Such problems usually have some optimization functions like minimizing the make span in job-shop scheduling or minimizing the overall sum of all routes in VRPs. These problems are known to be NP-complete and thus, in the general case, it is not tractable to perform an uninformed search in the solution space to find the optimal solution due to complexity.

As a trade-off between computational time and solution quality heuristic approaches have been introduced. These approaches do not exhaustively search the solution space but try to find a good solution by applying some domain specific heuristics or metaheuristics. Most of these metaheuristics start with a fixed set of parameters which does not change during runtime. Different metaheuristics are more suitable for different types of problems than others because of their character. Genetic Algorithms (GA), for example, are a fast optimization technique (Koza 1995). Depending on the binary construction of the different solutions and the easy way to mutate a solution, the genetic algorithm creates, in comparison to other metaheuristics, a new solution with a fast computation of the next generation. The disadvantage of this technique is that the best solution is found not as often as with other approaches (Koza 1995). The metaheuristic Particle Swarm Optimization (PSO) has been introduced by (Eberhart and Kennedy 1995). It also performs a randomized parallel search in the solution space. In comparison to PSO the computation of the next generation's individuals is faster in genetic algorithms but PSO focuses earlier to a specific minimum. Depending on the problem (i.e., the fitness landscape), different metaheuristics or varying parameters lead to better or worse efficiency in the optimization process. In standard metaheuristics

no dynamic changes (i.e., no adaptation) in parameter settings during the optimization process are intended. If an optimization starts, it will run until the termination criterion is reached. Finding the right termination criterion is another task when using metaheuristics. If the optimization terminates too early, a good solution may not be found. If the computation runs too long, computation time is wasted without relevant increase in the solution quality. We want to face these problems of missing dynamics during the optimization process and of finding adequate parameter sets for a combination of advantages of metaheuristics in this paper.

This paper is structured as follows. In section 2, we discuss how other existing frameworks address our challenges. In the third section we describe our framework and how a metaheuristic can be implemented. The fourth section provides an example how the framework can be used for optimization with PSO. In section 5 we discuss our approach and finally we take a brief look at the next steps or mode-specific options for our framework.

2 Related Work

Many different frameworks for using various metaheuristics are available. Most of them offer a lot of pre-defined functionality and metaheuristics like GA, Evolutionary Algorithm (SPEA2 and NSGA2), and Particle Swarm Optimization etc. The most common frameworks are OPT4J[1], JMETAL[2], EvA2[3] and JGAP[4]. These frameworks are libraries for *JAVA* and combine a lot of functionality for using metaheuristics. All types of metaheuristics can be initialized with different parameters and settings. To use the frameworks it is necessary to specify the problem in a specific language and implementation type. This translation is often difficult and a well known problem, because if there are abstract classes in the problem instance one has to write an en- and decoder for any problem types. After solving the problem of the translation into the right language, every optimization type can be configured and tested with the problem. During runtime of the optimization process there is no interaction between the optimization and the program which uses the optimization. Everything must be coordinated and configured before the start of the computation. The second handicap is that no dynamic adaptation of the optimization is possible. If disadvantageous parameters were chosen, one would have to wait until the end of the optimization process to change them. These approaches do not allow to use or combine intermediate results of different metaheuristics during runtime. Additionally, dynamic environmental changes influencing the search space or the fitness function cannot be taken into account until the optimization process reaches its termination criterion.

[1] http://opt4j.sourceforge.net

[2] http://jmetal.sourceforge.net

[3] http://www.ra.cs.uni-tuebingen.de/software/EvA2/

[4] http://jgap.sourceforge.net

The Paradiseo[5] framework provides distributed optimization on more than one central processing unit. Furthermore, it allows multi-objective problem types (Cahon et al. 2004). It has a straightforward implementation interface for adding new metaheuristics but there exists no interface for interrupting or observing the optimization process during runtime. It is possible to start the optimization and collect the data at the end of the optimization process. If one wanted to change parameters during runtime, he would have to stop and restart. Changing the fitness function leads to the loss of the current status of the optimization. The advantage of Paradiseo is that all integrated metaheuristics can be computed in parallel.

Finding the best parameters for a metaheuristic is a non trivial-task (Lee and El-Sharkawi 2008). Lots of techniques have been used to find the best parameter set for PSO (Tewolde et al. 2009). The disadvantage of the learning techniques is the iteration process where for every parameter set the whole optimization must be computed to get the information for the training set. Other approaches try to "understand" the fitness function first in a theoretical way and decide which parameters could be the best for the specific fitness landscape (Hutter et al. 2006).
Standard tuning methods are based on one or a fixed set of fitness functions and hence do not work well on other fitness functions especially if the fitness function changes during runtime. The real world shows us, that changes in the fitness function often happen. If a packet delivery driver starts with his route, he will compute an optimized route so that all packets are delivered in an efficient way. If he gets the information that one packet gets a new delivery address, he will have to recompute his way with this changed *fitness function*. Even if the fitness function changes only marginally during computation (e.g., packet A should be delivered to B and not C) it might be necessary to adapt the parameter set in order to maintain the result quality.

The dynamic exchange of solutions between different metaheuristics without losing information about the prior optimization is an open task and has never been done to the knowledge of the authors. The metaheuristics are usually run from the start to a termination criterion in order to get good results. But if the fitness function changes, changing the metaheuristic can be an advantage, e.g., because the fitness landscape might not be as cliffy as before. In this case it is better to change the metaheuristic without losing the solutions achieved so far.

With these problems of missing dynamics in mind we have developed a new agent-based approach (Lorion et al. 2009). In this work the framework is extended to acquire the missing dynamics in optimization processes.

[5] http://paradiseo.gforge.inria.fr

3 Framework for Dynamic Combination and Adaptation of Metaheuristics

In this section we present the architecture of our framework and describe how the optimization process can be controlled. We also give an example how new metaheuristics can be integrated, enhanced and controlled during runtime.

3.1 Architecture

The main structure of the framework is the agent construct. Figure 1 shows the abstract architecture of the framework agent. The advantage of an agent-based structure is the flexibility of the agent. The reasoning of the agent allows for steering the metaheuristic autonomously and enables problem-specific decision making. A social behaviour which affords the communication between agents is specified and lets the agents solve a problem together in a multi-agent system by exchanging their solutions (or information about the optimization process) of a problem. The agent contains a simple perceive-next-do structure (Wooldridge 2009). In dynamic environments the agent collects information with the sensors and via the communication device. The communication device allows communication with other agents and to share and distribute solutions. There exists a dependency between the problem to solve and which metaheuristic to use. To use PSO as metaheuristic, e.g., the problem must be a numerical optimization problem (usually in high dimensional spaces).

The reasoning component ("Next" in Figure 1) describes the connection between the agent behaviour and the metaheuristics. The behaviour of an agent is flexible and exchangeable. Only one simple interface has to be implemented to create new agent behaviours. Every behaviour contains several dynamic adaptation strategies for the optimization process and selects one of the provided strategies out of the strategy catalogue for the specific metaheuristic. These strategies are exchangeable during the optimization based on the offered flexibility of an agent structure and can change the parameter settings of the optimization. The behaviour observes the optimization and holds the current best solution at every time step. This solution can be compared with other solutions from other agents or other metaheuristics. Depending on the performance of the solution the selected strategy can decide to inject this solution to another metaheuristic during runtime or to change the parameters. The agent can start and stop the optimization process at every time step. With this feature every aspect of the optimization presets can be changed and adjusted w.r.t. new information, e.g., about the environment.

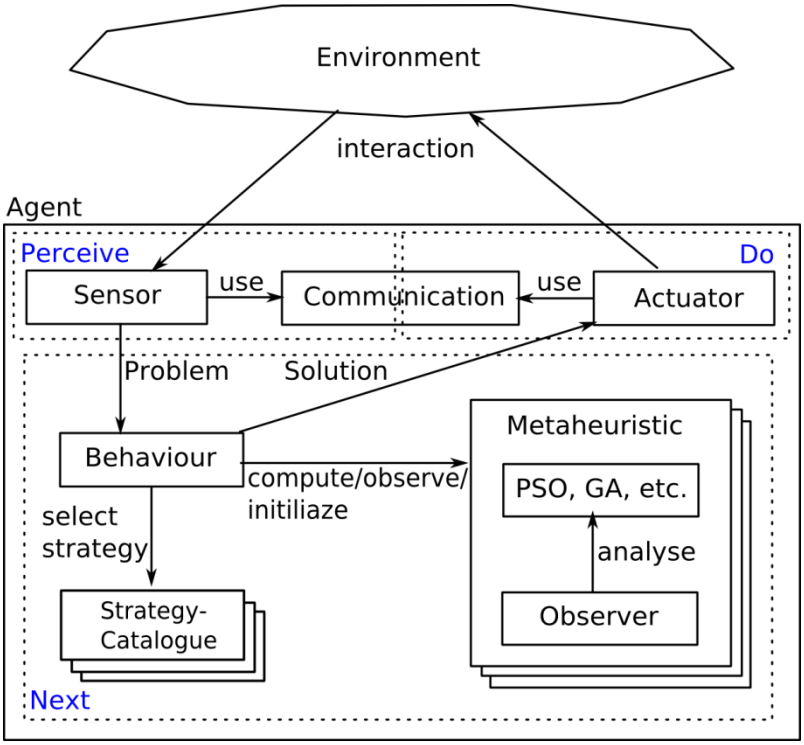

Figure 1: Abstract architecture of the agent

Depending on the problem every agent can choose between different metaheuristics and change the used metaheuristics during runtime (dynamic combination). More than one metaheuristic can be computed in parallel. The different metaheuristics can exchange and adapt solution of another metaheuristic within one agent as well. This dynamic combination of solutions of different metaheuristics provide a basis to solve problems with more than one type of metaheuristic. The agent starts the computation of the metaheuristic and observes the computation. The *Observer* provides data of the optimization process and a solution at every time step so anytime the agent could get any information about the process. This provided solution is comparable to solutions from other metaheuristics, if more than one is used, because every solution is stored as a metaheuristic independent *Entity*. This *Entity* interface gives the agent the ability to use different types of metaheuristics and to use hybrid optimization by exchanging the best solution between metaheuristics. Due to the communication abilities of agents, solutions can be exchanged between agents allowing cooperation within the multi-agent system. As shown in Figure 2, a metaheuristic contains a parameter set and a strategy. The *Strategy* holds a parameter set or an algorithm to adapt the parameters during the optimization process, i.e., applicable parameters of different metaheuristics can be handled here.

If the agent gets new information about the problem, e.g., that the optimization has to end soon, it could inject this information into the actual strategy or set a new strategy with the intention to get a better final result of the optimization.

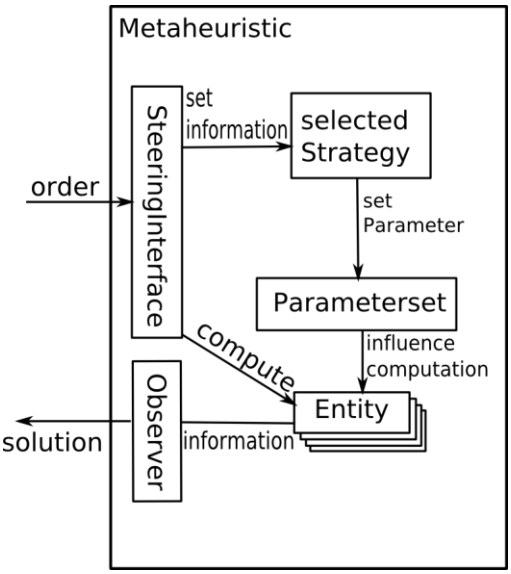

Figure 2: Interface of a metaheuristic

3.2 Starting an Optimization Process

One problem is which start parameters the metaheuristic should use. The easiest way is to set all start parameters to standard values from literature of the corresponding metaheuristic (in Particle Swarm Optimization: *w= 0.768, c1 = 1.4962, c2= 1.4962 (Clerc and Kennedy 2002)).* This is a standard way to initialize metaheuristics. In our framework every metaheuristic gets the strategy from the agent. It is possible to add a parameter optimizer which analyzes the fitness function and sets promising start parameters before the optimization process starts. This feature is configurable by the agent.

3.3 The Optimization Process

First every metaheuristic is initialized with a configuration set by the agent. The strategy is set and the fitness function is given by the agent. There are three options to start and run the optimization process. The agent can start the optimization normally and wait until it interrupts the process or the solution meets a termination criterion. The second option is to start the optimization only for one epoch. The agent can arbitrarily repeat this computation. The last option is to compute the optimization stepwise. The metaheuristic only computes the next fitness value

and then stops. Every time the agent stops the computation of the optimization, the metaheuristic is in a "waiting loop" where the actual state of the optimization is stored. In this state the agent can change and configure every aspect mentioned above. During the optimization process the strategy observes the fitness landscape and adapts the parameters accordingly. After every step or epoch of the optimization the metaheuristic yields the current best solution.

3.4 Implementing a New Metaheuristic

The framework is able to work with all population-based metaheuristics (like PSO, Genetic Algorithms, etc…). In order to implement new metaheuristics only a few interfaces have to be implemented. Every solution is an abstract *Entity*. This *Entity* has only a few methods which describe everything the agent and the metaheuristic has to know about the solution to share it with other metaheuristics, i.e. the fitness, the actual position in the function landscape and the actual properties (velocity, neighbours, etc…). To provide the current position and fitness it is necessary to build a decoder and encoder for different types of problems for every metaheuristic if one wants to share the solution between different types of metaheuristics. The framework provides a metaheuristic independent standard strategy which uses the standard parameter set of the corresponding metaheuristic. Additionally, the standard strategy can perform simple modifications, like increasing or decreasing a parameter value. Every metaheuristic should provide – independent of the parameter adaptation strategies – a standard parameter set that could be used if no parameter sets are given.

The last step on the way to a new metaheuristic is the Observer. This interface only contains five important functions: two functions get the fitness values and the collected data about the optimization process. This is only important for the statistics at the end of the run. Two functions are for the classifiers if the strategy uses a learning technique for parameter adaptation. The main function is the analyze function which provides the actual state of the optimization. With these functions the Observer provides all the data about the optimization for the agent.

3.5 Using a Classifier

To analyze the optimization process and to set the right parameters at the right time machine learning techniques should be applied. To provide different techniques, this framework allows every observer to use different kinds of machine learning techniques like *J4.8 tree (*an implementation of *C4.5)* and *neuronal networks*. These classifiers are implemented in the *WEKA machine learning toolkit*[6]. To use these techniques as classifiers every classifier has to be trained with the data provided by the observer. Different problems have to be solved to get the training

[6] http://www.cs.waikato.ac.nz/ml/weka/

data and to be independent from the fitness landscapes of single problems. If the classifier is trained, the observer can use it to get the actual state of the optimization and help the strategy to set the right next parameter sets for the metaheuristic.

3.6 Statistical Visualization

Every agent holds a statistic class which provides an output format for *R-Project*[7] to visualize the optimization run or to perform statistical significance tests (e.g., t-test, ANOVA, etc.) to check if an optimization run is better than another.

4 Application to PSO

We have implemented the framework as described in the previous section. In order to demonstrate the usability of the framework and the possibilities using the framework for research we adapt the Particle Swarm Optimization and test how a dynamic switch of the parameters influences the optimization process. For dynamic adaptation of the PSO, we introduce a number of swarm properties that can be used for selecting an adequate parameter set. The basic idea is to learn an adaptation strategy by identifying good parameter sets w. r. t. the swarm properties at a current time step in the optimization process.

4.1 Integrating a Metaheuristic

To describe how we designed a PSO with our framework we simply show how the design of the metaheuristic is structured. The interface *Metaheuristic* is implemented by the class *Swarm* which holds the whole swarm. To compute the swarm a few functions have to be implemented which either compute a whole generation or simply one particle after another. Every particle is implemented as an *Entity*. The strategy catalogue contains different strategies with fixed strong parameter sets. One additional strategy includes a classifier which is trained by examples with results of previous PSO runs with different parameters. The classifier selects the parameter set to be used based on a number of swarm properties and thus implements an adaptive strategy for PSO. The properties which describe the swarm and the optimization process are described in the Observer. The first three properties address the velocity of the particle (see Figure 3). The observer computes the average speed (see Fig 3.a) of the particles during the epoch, as well as the highest and lowest speed of the swarm (see Fig 3.b). Other properties are the direction of the swarm and the particles. Is the swarm focussing or spreading and to which direction does the average direction of the velocities of every particle

[7] http://www.r-project.org/

point (see Fig 3.c). Focussing means that the swarm is moving towards a minimum and slowing the speed down.

The last properties are about the positions of the particles. The observer computes the average distance to the best point of the swarm (see Figure 4.a), the largest distance between two particles (see Fig 4.b) in the swarm, and the convex hull of all particles (see Fig 4.c).

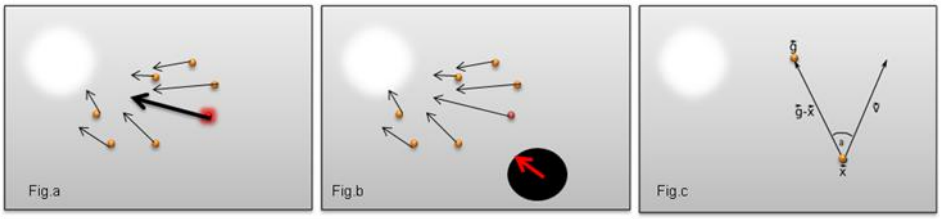

Figure 3: Velocity properties

To avoid the problems of different functions, the fitness is not taken into account as a property as different problems can have different scales of the fitness landscape.

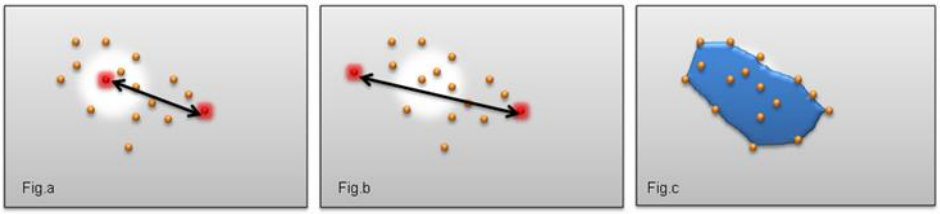

Figure 4: Position properties

4.2 First Experiments with this Framework Using a PSO

As first experiments we want to learn strategies for parameter adaptation. To solve this problem, we use a supervised learning technique to build up a classifier. To build the training set for this classifier we need different strategies to collect data for the different parameter sets. In Figure 5 we show how we collect and use training data to build the classifier.

As described in Section 3 it is possible to hold more than one metaheuristic per agent. In our example we use 20 PSOs with different parameters. As a first step we initialize one swarm and copy the particles (*entities*) to the other swarms, i.e.,. every swarm starts at the same position. Then the agent starts the computation of all metaheuristics for 20 epochs. The agent stops the computation and evaluates the latest fitness values from every swarm. The fitness combined with the properties of the swarm is provided and stored by the observer of every swarm.. After collecting the data the agent compares the data and sets the actual best fit-

ness as best strategy parameter for the collected properties of the swarm in the previous collecting phase. After that all best swarm's particles are copied to the other swarms and replace all particles so that all swarms are identical. Repeating this step for a number of rounds lets all collected properties get a best parameter set and completes the training set (i.e., swarm properties and the identified parameter set with highest fitness for this situation).

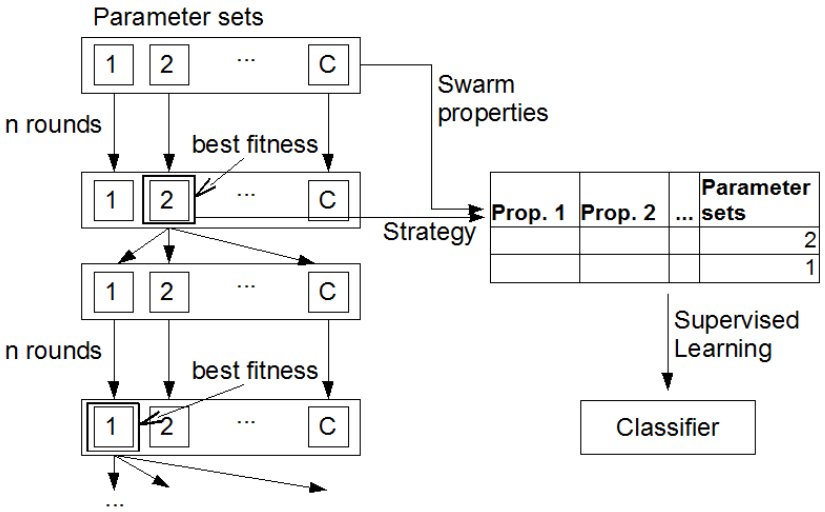

Figure 5: Generation of training data for learning an adaptation strategy

These properties combined with the parameters showing the best fitness after 20 epochs are used as a training set for the classifier. The learned classifier is later used for dynamic parameter switching in the adaptation strategy.

4.3 Applying the Classifier as an Adaptation Strategy

The learned classifier is used for dynamic parameter switching in a strategy. This strategy automatically switches to the learned parameter set depending on the observer data of the current situation. Finally we compare this dynamic parameter switching PSO to 20 normal PSOs using a standard parameter set and choose after termination the best swarm as competitor. The result shows us that more swarms switching to the swarm with the best parameter set are more efficient than one swarm *switching to the best* parameter set. This can be explained by the higher amount of computational power. If we compare the adaptive strategy to only one standard PSO we get the same result depending on the randomization during the optimization process. This probably depends on the number of swarms because independent of the chosen parameter set a higher number of swarms has a bigger

change to reach a better fitness. A next step is to find another strategy and to compare the different parameter sets on different fitness functions.

In order to analyse the potential of different parameter sets, we selected different kappa values of the *constriction update function* (Eberhart and Shi 2000) and applied them to different functions. The visualization of the mean fitness values over time using the statistics class can be seen in Figure 6. Every colour represents a different kappa value. The red curve has the best solution in the end but does not show fast optimization in the beginning. In contrast, the blue curve has a strong draft down but then the swarm focuses too early. As we see with this framework it is possible to analyse efficiently the metaheuristic and find new strategies to gain a better result with one optimization run.

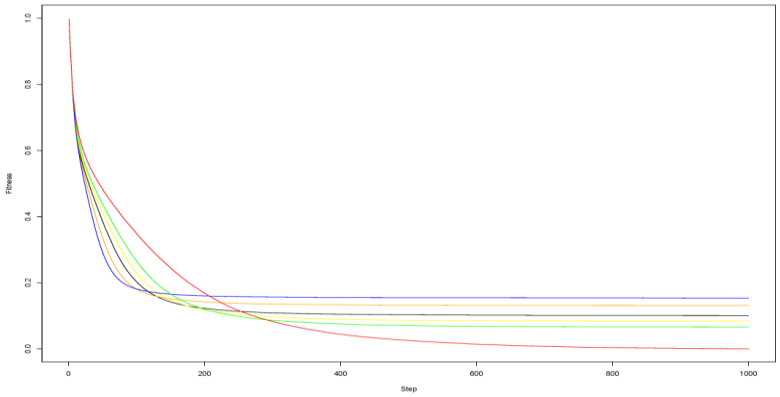

Figure 6: Different kappa values (average over five different functions)

5 Discussion

In this work we presented a new approach for analysing and using metaheuristics in a more efficient and easier way. The extension to new metaheuristics is simple through the few interfaces that have to be implemented. The underlying agent structure guarantees flexibility and autonomous computation of metaheuristics in a wide field of technology. We are at the beginning of the development of this framework and some features depend on our own systems in the current status. The agents communicates with an own language and have no official agent communication language. We avoid to use an agent framework like *JADE* (Bellifemine et al. 2000) in the first step because we want to build an easy agent structure which can be adapted and reused easily on all systems. The first experiments with this framework have been done and demonstrated the flexibility as well as the possibility to change parameters dynamically in a metaheuristic.

6 Future Work

The next steps are to make our agent compatible to *JADE*. This implicates that the communication of the agents change to the ACL (Poslad et al. 2000). With this change it is possible to build up a multiagent system with an underlying optimized *JADE* structure. Being compliant to ACL allows for interacting with other agents following this standard as well.

Furthermore we want to expand our research on dynamic parameter adaptation during the optimization process to find out whether different parameter sets gain a better solution in efficient time as a standard metaheuristic run and if it is possible to find the best strategy autonomously for different sets of problems.

References

Bellifemine, F., Poggi, A., Rimassa, G., and Turci, P. (2000). "An Object Oriented Framework to Realize Agent Systems"*Proceedings of WOA 2000 Workshop*. City: Parma, pp. 52-57.

Cahon, S., Melab, N., and Talbi, E. G. (2004). "ParadisEO: A framework for the reusable design of parallel and distributed metaheuristics." *Journal of Heuristics*, 10(3), 357 - 380.

Clerc, M., and Kennedy, J. (2002). "The particle swarm-explosion, stability, and convergence in amultidimensional complex space." *IEEE transactions on Evolutionary Computation, 6(1), 58 - 73*.

Eberhart, R., and Kennedy, J. (1995). "A New Optimizer using Part Swarm Theory." *Proceedings of the SixthInternational Symposium on Micro Maschine and Human Science*, 39-43.

Eberhart, R. C., and Shi, Y. (2000). "Comparing inertia weights and constriction factors in particle swarm optimization." *Evolutionary Computation, 2000. Proceedings of the 2000 Congress on*, 1, 84-88.

Hutter, F., Hamadi, Y., Hoos, H. H., and Leyton-Brown, K. (2006). "Performance Prediction and Automated Tuning of Randomized and Parametric Algorithms", *Principles and Practice of Constraint Programming (CP'06)*. pp. 213-228.

Koza, J. R. (1995). "Survey of genetic algorithms and genetic programming"*WESCON/'95. Conference record. 'Microelectronics Communications Technology Producing Quality Products Mobile and Portable Power Emerging Technologies*. City: Dept. of Comput. Sci., Stanford Univ., CA: San Francisco.

Lee, K. Y., and El-Sharkawi, M. A. (2008). *Modern Heuristic Optimization Techniques*: IEEE Press.

Lenstra, J. K., and Kan, A. H. G. R. (2006). "Complexity of vehicle routing and scheduling problems." *Networks*, 11, 221 - 227.

Lorion, Y., Bogon, T., Timm, I. J., and Drobnik, O. "An Agent Based Parallel Particle Swarm Optimization - APPSO." *Presented at Swarm Intelligence Symposium SIS 2009*, Nashville, TN.

Poslad, S., Buckle, P., and Hadingham, R. "The FIPA-OS agent platform: Open source for open standards."

Tewolde, G. S., Hanna, D. M., and Haskell, R. E. "Enhancing performance of PSO with automatic parameter tuning technique." *Presented at IEEE Swarm Intelligence Symposium. SIS'09*.

Wooldridge, M. (2009). *An introduction to multiagent systems*, Chichester: Wiley.