BINMETA: a new Java package for meta-heuristic searches

Antonio Mucherino

BINMETA: a new Java package for meta-heuristic searches

Antonio Mucherino

IRISA, University of Rennes 1, Rennes, France.
Email: antonio.mucherino@irisa.fr

Abstract. We present a new Java package, named BINMETA, for the development and the study of meta-heuristic searches for global optimization. The solution space for our optimization problems is based on a discrete representation, but it does not restrict to combinatorial problems, for every representation on computer machines finally reduces to a sequence of bits. We focus on general purpose meta-heuristics, which are not tailored to any specific subclass of problems. Although we are aware that this is not the first attempt to develop one unique tool implementing more than one meta-heuristic search, we are motivated by the following three main research lines on meta-heuristics. First, we plan to collect several implementations of meta-heuristic searches, developed by several programmers under the common interface of the package, where a particular attention is given to the common components of the various meta-heuristics. Second, the discrete representation for the solutions that we employ allows the user to perform a preliminary study on the degrees of freedom that is likely to give a positive impact on the performance of the meta-heuristic searches. Third, the choice of Java as a programming language is motivated by its flexibility and the use of a high-level objective-oriented paradigm. Finally, an important point in the development of BINMETA is that a meta-heuristic search implemented in the package can also be seen as an optimization problem, where its parameters play the role decision variables.

1 Introduction

Meta-heuristic searches are general purpose methods for global optimization [10, 14], which are generally inspired by the observation of the nature, or of animal behavior, including the social behavior of human beings [6]. Among the most "famous" meta-heuristics we can cite the family of Generic Algorithms [13], and swarm intelligence approaches such as Ant Colony Optimization [4]. The BINMETA package that we present for the first time in this article is intended for collecting the implementations of several of the meta-heuristic searches that have been proposed in the scientific literature in recent years.

We plan our BINMETA package to be the instrument for a wide comparison among different methods and strategies implemented in various meta-heuristic searches, that can potentially be complementary and complete each other under the common interface provided by the package. One main idea is to keep the implementations rather "simple", while remaining effective, and very modular, so that the involvement of other

researchers in the development of the package is encouraged. In other words, the main approach for the development of this Java package is the so-called incremental build approach [11]. The initial versions of the package have already been deposited on the GitHub[1], in a public repository. We make reference in the following to the 13^{th} commit (code: `2007aa2`) to this repository.

We make the choice to represent the solution space of all considered optimization problems in a combinatorial space, so that the implemented meta-heuristic searches can work in a discrete space. A discrete representation is naturally accommodated for combinatorial problems, but we do not aim to limit, with this choice for the representation of the solutions, our package to solely combinatorial problems. In fact, the solutions to every optimization problem need to admit a suitable discrete representation (in practice, a representation as a bit string) in order to be treated by digital machines. Moreover, we pay particular attention to having *optimal* discrete representations for the solutions of our optimization problems, where we intend optimality in terms of the degrees of freedom for the solutions.

BINMETA is developed in Java. The choice of a high-level and object-oriented programming language comes first of all for encouraging a wide collaboration in its development (as already mentioned above). Moreover, Java allows us to develop the different parts of the package (optimization problems, meta-heuristic searches) under a common interface, where every part can be easily related to the others. One initial advantage of this flexibility is given by the fact that meta-heuristic searches are implemented so that they are compatible with the interface describing the objective functions of the optimization problems. The selection of the "optimal" parameters for a given meta-heuristic search for the solution of a given optimization problem can in this way be seen as another optimization problem, where the parameters of the meta-heuristic search play the role of decision variables. The solution to this parameter tuning problem can therefore be attempted with our Java package.

We point out that this is not the first software project on meta-heuristic searches. There exist in fact several implementations of single (or of particular subgroups of) meta-heuristic searches; the more one meta-heuristic search is studied and used in the scientific community, the more there are implementations available. Other software tools may not focus on meta-heuristics but they may contain some of their implementations for tackling some specific problems (examples can be found in [14]). Finally, other software projects collecting several meta-heuristics include the software tool PARADISEO [2] (written in C++), as well as the software tool JMETAL [5], written in Java and mainly focused on multi-objective optimization. To the best of our knowledge, apart from the main features already pointed out above (data representation and optimization, organization of class interfaces), BINMETA is the first software project whose development strictly follows an incremental programming approach, and it encourages the participation of several developers.

The rest of the paper is organized as follows. In Section 2, we will give the details of the binary representations that will be employed for the possible solutions to our optimization problems. Section 3 will present some of the optimization problems that are currently available in the BINMETA package, while Section 4 will describe the very first

[1] `https://github.com/mucherino/binMeta`

meta-heuristic searches that have been implemented in the package. Finally, Section 5 will present some preliminary experiments, and Section 6 will conclude the paper.

2 Representation of the solutions

Some meta-heuristic searches were originally conceived and developed to admit a discrete representation of the solutions. For example, in Ant Colony Optimization (ACO) [1], the solutions are vertices of a given graph, where the involved *ants* move by stepping from one vertex to another where an edge exists between the two. The search space is therefore discrete and admits representations of the entire set of potential solutions in a combinatorial space. Other meta-heuristics, instead, are strongly based on continuous representations of the solutions: one example is the recent Spiral Optimization meta-heuristics [15], where the constructed *spirals* admit continuous representations in Euclidean spaces. Even if initially conceived for a discrete or continuous representation of the solutions, the implementation adaptations of meta-heuristic searches from one to the other representation have been attempted in previous works (see for example [3]).

In our Java package, the choice to represent the solutions to all optimization problems in a combinatorial space comes from the observation that all suitable representations on a computer machine *need* to be discrete (and finite). Even continuous problems with real-valued variables, for example, need to have their variables represented on computer machines in a binary format, which essentially corresponds to a well-formatted "bit string" (the *floating-point* representation is for example the proper format for the real-valued variables). Instead of using these generally provided standard data types, BINMETA gives the user the possibility to develop ad-hoc and specific binary representation for the involved variables. If a given variable can take up to four different values, for example, then 2 bits are sufficient to represent it. If solutions of the problem at hand are sets of variables that can each take up to four values, these solutions can be represented in binary format as a *unique* bit string concatenating the 2 bits necessary for the representation of each variable.

We point out that the use of one unique bit string for the representation of the solutions is much more efficient than considering arrays of single variables. In fact, in the previous example, we would have an array of variables of type `byte`, each covering 8 bits in the computer memory. However, only four values can be taken by each variable stored in the byte, and therefore 6 out of the 8 bits are actually not used. Moreover, if they were used, they would correspond to values that are not feasible for these variables, so that constraints need to be included and verified for a correct use of the array of bytes. These two issues are immediately overcome when employing one unique bit string representation.

3 The first optimization problems

We give in this section some examples of optimization problems that are currently implemented in our Java package BINMETA. Some of them are simple problems, which

were included in the package with the only aim to performing simple tests on the meta-heuristic searches, before attempting the solution of harder problems. Some of our optimization problems are classical problems arising in the field of Operational Research [18]. Our current package version includes the SUBSET SUM PROBLEM, the NUMBER PARTITION PROBLEM and the KNAPSACK PROBLEM.

In the following sections, we will focus instead on two other optimization problems that we have included in the package mainly for testing and presentation purposes (Sections 3.1 and 3.2). Finally, in Section 3.3, we will briefly discuss the possibility to solve optimization problems where the objective function provides, for a given set of parameters' values, the performance of the implemented meta-heuristic searches (that we will discuss in Section 4).

3.1 The Pi objective

This objective is related to an optimization problem that can be considered as an *easy* problem: we decided to describe it in details for giving an example of simple problem with a set of real-valued variables that can be encoded as a unique bit string having fixed length.

The basic idea is to find the positions on a unit circle (lying in the two-dimensional Euclidean space) of n points x_i such that the objective

$$Pi(\{x_1, x_2, \ldots, x_n\}) = \sum_{i=1}^{n} ||x_i - x_{i+1}||$$

is maximized, where it is supposed that x_{i+1} coincides with x_1, and where $||\cdot||$ represents the Euclidean norm. Finding the set of n points that maximizes the value of the Pi function is equivalent to improving the approximation of the number π that is obtained.

As mentioned above, this is not a hard problem: if every point x_i is constrained to take positions in a portion of circle that is in size proportional to $1/n$, then the optimal solution can be simply identified by local optimization. However, we did not impose this constraint in the implementation of the objective in our BINMETA package.

The precision of the point positions, in terms of number of bits used their representation, can naturally have an important impact on the obtained approximation of π. Since every x_i belongs to the unit circle, we can represent it with only one real value (a vector angle); a discrete set of possible values for this angle can then be represented by a bit sub-string having a predefined length (which will correspond to the precision of the representation). Finally, one possible solution to the problem can be represented by the bit string concatenating all sub-strings related to the points x_i.

3.2 The Fermat objective

Fermat's last theorem states that no solutions for the equation $x^n + y^n = z^n$ exist when x, y and z are positive integers larger than 1, when the value of n is fixed and is greater than 2. This theorem was proved in 1995 by Sir. Andrew Wiles [17], about 350 years after the formulation of the theorem by Pierre de Fermat. We consider the following objective:

$$F(x,y,z) = |z^n - x^n - y^n|,$$

which basically measures the violation of the Fermat equation for a fixed value of n, and for three possible integers x, y and z. If we attempt the minimization of F when $n = 2$, we know that there exist solutions where the value of the function can be zero. However, if we could find a triplet $(\hat{x}, \hat{y}, \hat{z})$ for which the value of this objective is zero when $n > 2$, this would correspond to find a counterexample to Fermat's last theorem.

Differently from the problem in the previous section, optimizing the Fermat objective is NP-hard. This can be easily proved by noticing that the problem is equivalent to a SUBSET SUM instance where, in the set of all positive integers of the type x^n smaller than a given upper bound, and which also contains their opposites $-x^n$, it is necessary to verify whether there exist a subset whose element sum is zero, with the additional constraint that the cardinality of the subset must be 3. Two copies of x^n may be included in the initial set to consider equations where $x = y$.

This objective allows us to point out the importance of the choice of the binary representation (the final bit string) for the possible solutions to the problem. We can remark, first of all, that by inverting x and y in the original equation, the solution does not change. Therefore, half of the possible solutions (which are symmetric w.r.t. other solutions) can be removed by imposing the constraint $x \leq y$. Moreover, we can also remark that the values that z can take are also constrained by the values assigned to the other two integers.

Once the number of bits for the integer representations is selected (equivalent to giving an upper bound on the values of the integers), the final bit string representing a possible solution to the Fermat problem is the concatenation of the three sub-strings encoding the three integers x, y and z. However, for the reasons given above, only x is represented in absolute value, while the difference $y - x$ is encoded by the second sub-string, and the difference $z - y$ is encoded by the third sub-string.

3.3 Meta-heuristic searches as objective functions

Recent works have been focusing on the problem of setting up automatically the parameters involved in meta-heuristic searches [12]. In our Java package, a special class of optimization problems is given by the set of implemented meta-heuristic searches, where the objective functions provide a measure of their performance when invoked to solve a certain problem. The decision variables are in this case the set of parameters that are used to invoke the meta-heuristic search, represented as a bit string in our implementations. The main idea is to find the parameters that allow the search to achieve the best performance when dealing with a specific problem (or a particular class of problems). In BINMETA, the meta-heuristic search A may be used for optimizing the parameters of the meta-heuristic search B in the attempt to solve the optimization problem C. The situation where A = B is also feasible from a technical point of view in our Java package.

4 The first meta-heuristic searches

We focus in this section on one of the first meta-heuristic searches that have been implemented in our Java package BINMETA, the Wolf Search meta-heuristics (see Sec-

tion 4.1). Notice however that a Random Walk meta-heuristics has also been implemented, as well as a local optimization procedure inspired by the gradient descent method but acting directly on the binary representations of our solutions.

4.1 Wolf Search

Wolves can hunt individually or in a group. These groups of wolves are however local, they do not comprise the entire set of wolves in a given area. A wolf moving individually in search for food would alternate its efforts in catching a new prey by itself, and in verifying whether a near wolf has been luckier in its searches and eventually join it. The risk for a threat, given by the presence of other predators, can potentially make the wolf decide to leave its current search, in order to escape in a safer area. We implemented the Wolf Search as described in [16], because well adapted to the spirit of our Java package; we remark that a more sophisticated meta-heuristic search, that is also based on the hunting behavior of wolves, was subsequently proposed in [7].

The Wolf Search meta-heuristics is based on the following three main steps. First of all, every wolf in the group attempts to perform an individual move in the search space, with the aim of finding better hunting conditions (more food, more preys, represented by a solution with better objective function values). The new solution is accepted only if it does improve the objective function value, and only if it was not recently considered (for this second criterium, we suppose that the wolves are equipped with an ephemeral memory containing previously explored solutions). In our implementation, the wolf initially looks randomly in its range of vision, and then tries to improve its current solution by performing a local search.

If the individual search does not lead to any new better solutions, then the wolf looks around itself to verify whether other wolves had been luckier in the search. To simulate locality, only the wolves positioned in solutions that are close in distance to the current wolf's solution are taken into consideration. The current wolf can therefore decide to join another wolf when this other one is in its range of vision, and if this other wolf is currently carrying a better solution. Joining another wolf corresponds to standing in the solution it is carrying, or in a very near solution.

Finally, every wolf has a given risk probability for a threat, which will imply a random movement in the search space, but limited to the wolf's range of vision. This movement is completely independent from the current value of the objective function; since the wolf behavior may make the wolves escape from good quality solutions, these solutions are not included in the ephemeral memory when escaping, in a way that these wolves may come back exploring the same part of the search space when the threat has disappeared.

5 Preliminary experiments

We propose some preliminary computational experiments, where the Random Walk and Wolf Search meta-heuristics (see Section 4.1) are employed for solving the optimization problems represented by the objective function Pi (see Section 3.1) and Fermat (see Section 3.2).

	n	*#digits*	*#bits*	Random Walk	Wolf Search
Pi	10	3	30	-3.0880	-3.0901
	25	4	100	-3.1192	-3.1325
	50	5	250	-3.1202	-3.1321

	n	*#digits*	*#bits*	Random Walk	Wolf Search
Fermat	2	10	30	1.0	0.0
	3	12	36	16353.0	1.0
	3	15	45	1849940.0	2.0

Table 1. Every meta-heuristic run for 1 second, on a standard computer laptop. Since our meta-heuristics are implemented to solve minimization problems, a negative sign is assigned to every evaluation of the Pi objective.

On the left-hand side of Table 1, we report some solutions found when optimizing the Pi objective. The value of n indicates the number of points on the unit circle, while *#digits* indicates the number of bits used for the representation of their corresponding angle. The total number of bits (*#bits*) is a consequence of the previous two values. The experiments show that better results can be obtained with larger values for both n and *#digits*. As expected, Wolf Search performs better than Random Walk.

On the right-hand side of Table 1, some experiments concerning the Fermat objective are proposed. In this case, n is the exponent in the Fermat equation, while *#digits* is the number of bits used in the representation of each integer x, y and z (and therefore the total number of bits corresponds to 3 times this value). When $n = 2$, Wolf Search is able to find a solution for which the objective is zero: in this solution, $x = 3$, $y = 5$ and $z = 10$. In the other two experiments, solutions to the equation are not supposed to exist.

6 Conclusions

We introduced the new BINMETA package, a Java package for the implementation and the study, under a common interface, of several meta-heuristic searches for global optimization. Future works will consist in extending the number of objectives, as well as the number of meta-heuristic searches, that are implemented, so that BINMETA can become one of the main references for the study of meta-heuristic searches. We also plan to introduce the concept of meta-heuristic *environment* in our package, as described in [9]. Finally, we point out again the importance in choosing the binary representation for the solutions: this is an important point that was studied for example in [8] for the distance geometry problem, which we plan to include in the near future in our package.

Acknowledgments

Throughout the entire article, the reader may have noticed that the plural form is employed even if there is only one author. This author actually needs to thank the collaboration of some Master students that worked on this software package in the framework of course projects. The identity of the students that gave the most important contributions appear (in different forms) in the source files (see GitHub repository).

This work is partially supported by the international project MULTIBIOSTRUCT funded by the ANR French funding agency (ANR-19-CE45-0019).

References

1. V. Atanassova, S. Fidanova, I. Popchev, P. Chountas, *Generalized Nets, ACO-Algorithms and Genetic Algorithm*. In: "Monte Carlo Methods and Applications", K.K. Sabelfeld, I. Dimov, De Gruyter, 39–46, 2012.
2. S. Cahon, N. Melab, E-G. Talbi, *ParadisEO: A Framework for the Reusable Design of Parallel and Distributed Metaheuristics*, Journal of Heuristics **10**, 357–380, 2004.
3. B. Crawford, R. Soto, G. Astorga, J. García, C. Castro, F. Paredes, *Putting Continuous Metaheuristics to Work in Binary Search Spaces*, Complexity **2017**, article ID8404231, 19 pages, 2017.
4. M. Dorigo, M. Birattari, *Ant Colony Optimization*. In: "Encyclopedia of Machine Learning", C. Sammut, G.I. Webb (Eds.), Springer, 36–39, 2010.
5. J.J. Durillo, A.J. Nebro, *jMetal: a Java Framework for Multi-Objective Optimization*, Advances in Engineering Software **42**, 760–771, 2011.
6. I. Fister Jr, X-S. Yang, I. Fister, J. Brest, D. Fister, *A Brief Review of Nature-Inspired Algorithms for Optimization*, Elektrotehniski Vestnik **80**(3), 1–7, 2013.
7. S. Mirjalili, S.M. Mirjalili, A. Lewis, *Grey Wolf Optimizer*, Advances in Engineering Software **69**, 46–61, 2014.
8. A. Mucherino, *An Analysis on the Degrees of Freedom of Binary Representations for Solutions to Discretizable Distance Geometry Problems*. To appear in "Recent Advances in Computational Optimization", S. Fidanova (Ed.), Studies in Computational Intelligence, 2021.
9. A. Mucherino, S. Fidanova, M. Ganzha, *Ant Colony Optimization with Environment Changes: an Application to GPS Surveying*, IEEE Conference Proceedings, Federated Conference on Computer Science and Information Systems (FedCSIS15), Workshop on Computational Optimization (WCO15), Lodz, Poland, 495–500, 2015.
10. A. Mucherino, O. Seref, *Modeling and Solving Real Life Global Optimization Problems with Meta-Heuristic Methods*. In: "Advances in Modeling Agricultural Systems", Springer Optimization and Its Applications **25**, P.J. Papajorgji, P.M. Pardalos (Eds.), 403–420, 2008.
11. R.S. Pressman, B.R. Maxim, *Software Engineering: A Practitioner's Approach*. McGraw-Hill Education, 9^{th} edition, 704 pages, 2019.
12. D. Pukhkaiev, Y. Semendiak, S. Götz and U. Aßmann, *Combined Selection and Parameter Control of Meta-heuristics*, IEEE Conference Proceedings, Symposium Series on Computational Intelligence (SSCI20), Canberra, Australia, 3125–3132, 2020.
13. S. Sivanandam S. Deepa, *Introduction to Genetic Algorithms*, Springer, Berlin, Heidelberg, 442 pages, 2008.
14. K. Sörensen, F. Glover, *Metaheuristics*, Encyclopedia of Operations Research and Management Science **62**, 960–970, 2013.
15. K. Tamura, K. Yasuda, *Spiral Optimization Algorithm Using Periodic Descent Directions*, SICE Journal of Control, Measurement, and System Integration **9**(3), 134–143, 2016.
16. R. Tang, S. Fong, X.S. Yang, S. Deb, *Wolf Search Algorithm with Ephemeral Memory*, IEEE Proceedings, 7^{th} International Conference on Digital Information Management (ICDIM 2012), Macau, 165–172, 2012.
17. A. Wiles, *Modular Elliptic Curves and Fermat's Last Theorem*, Annals of Mathematics **141**(3), 443–551, 1995.
18. G.J. Woeginger, *Exact Algorithms for NP-hard Problems: A Survey*. In: "Combinatorial Optimization – Eureka, You Shrink!", Springer, Berlin, Heidelberg, 185–207, 2003.