Metaheuristics in Implementation Challenges—some findings you might find helpful

Martin Josef Geiger
University of the Federal Armed Forces Hamburg, Germany
m.j.geiger@hsu-hh.de

Abstract
We are fond of optimization/implementation challenges. First of all for one reason: they provide us researchers with a testbed for applying our methods, skills, and ideas, and come with impartial verification of the results by a third party, i. e. the challenge organizers. Moreover, the development of the solution approaches must be completed within a finite, typically short time horizon, and important factor when judging on the 'practical applicability' of a solution method. Even better, the problems are not chosen by the participants themselves. When putting everything together, some meaningful comparison should arise.

We may suspect that metaheuristics can successfully be applied in such scenarios. While this is, from our experience, true, their actual application is not so easy. This paper therefore collects some experiences with actual implementations for different optimization challenges that you may find helpful.

Keywords: Metaheuristic Design, Variable Neighborhood Search, Iterated Local Search, Implementation Challenges.

1 Introduction

"Metaheuristics are among the most prominent and successful techniques to solve a large amount of complex and computationally hard combinatorial and numerical optimization problems arising in human activities, such as economics, industry, or engineering. Metaheuristics can be seen as general algorithmic frameworks that require a relatively small number of modifications to be adapted to tackle a specific problem. They constitute a highly diverse family of optimization algorithms, that all have their individual properties, strengths and issues." [2]

While the above given introduction is rather broad, this paper gives a more focused view on the use of metaheuristics for solving optimization problems. Here, we concentrate on our lessons learned from implementation challenges, namely the International Timetabling Competition 2007 (ITC2007), the Nurse Rostering Competition 2010 (NRC2010), the Kaggle "Traveling Santa" Problem (Kaggle2012), the MISTA 2013 Challenge (MISTA2013), the VeRoLog Solver Challenge 2014 (VeRoLog2014; joint work with S. Huber [8]), the EURO/ROADEF Challenge 2014 (ROADEF2014; joint work with S. Huber, S. Langton, M. Leschik, C. Lindorf, U. Tüshaus), the VeRoLog Solver Challenge 2015 (VeRoLog2015), and the Kaggle "'Santa's Stolen Sleigh" Problem (Kaggle2015). Having developed source code for those competitions, some experiences have been gathered that we hope to be of interest to the reader. The subsequent sections follow the questions raised by the EU/ME 2016 Workshop, and present answers to them from our personal perspective.

2 What is a good metaheuristic design?

To the best of our knowledge, there is no universal definition of "good metaheuristic design". From a publish-or-perish-point-of-view, any metaheuristic that leads to a publication can be described as good/successful, although we do not adopt this definition for our work. From a theoretical perspective, convergence of the proposed metaheuristic towards the global optimum is sometimes investigated [6]. While this is of general interest, and while parameter settings can be derived from theoretical investigations, solving a problem under (tight) running time restrictions is different. For our purposes, "good metaheuristic design" rests on two pillars:

1. Quality of the obtained solutions under the given running time restrictions.
   In this context, quality of solutions is measured relative to the results of other approaches, and this can be complemented with lower bound calculations. Typically, several but few instances/data sets are used for testing (from around 10 to around 30, in some cases only 1 (Kaggle2012, Kaggle2015)). This means that a fair amount of parameter tuning can be done, but also implies that the proposed solution will/must be narrowed down to the specific problem at hand. Note that when publishing the results, this can turn out to be problematic.
   Creating own instances, while keeping the key characteristics of the original data, is recommended. This enables experiments on other datasets and helps when looking for unexpected weaknesses or (logical) programming errors.

It is also a good idea to study up on parallel programming: In the recent Kaggle2015, the winning approach used a 24-core computer, and also other recent competitions allow the usage of multi-core-architectures (typically 2–4 (MISTA2013, ROADEF2014, VeRoLog2014, VeRoLog2015)).

2. Possibility of fast adapting and modifying the implementation.
(i) One the one hand, this is an argument for the usage of metaheuristic frameworks (Section 3). Such concepts promise a fast adaptation of different techniques, such as local search operators, acceptance principles, etc. (ii) On the other hand, and typically in the final phase of an implementation challenge, runtime efficiency of the code must be considered. This might work with metaheuristic frameworks, given some detailed knowledge of the source code and the underlying data structures. In our experience, the former aspect is outweighed by the latter: code efficiency is paramount.

Note that there are problems for which we believe that a metaheuristic alone will not give competitive results. This is the case when e. g. Linear Programming or another exact optimization approach can be put to work, and it applies to several cases of otherwise 'hard' optimization problems (timetabling, rostering, scheduling, knapsack/cutting/packing). As an example, our Variable Neighborhood Search for NRC2010 did not even qualify for the finals, despite solving 7 instances to optimality and proving another best-known solution (out of 20 datasets): the winning approach used a heuristic in combination with mathematical programming [16]. In the ITC2007, the winning approach employed Constraint Programming, and surpassed our Threshold Accepting algorithm [5].

3 Which metaheuristic framework to use?

Metaheuristic frameworks are not a recent phenomenon. In fact, they became a popular area of research in the late 90s of the past century [1, 12, 17], and several have been developed in the following years [3, 4], some of which turned into commercial software packages or freely available software [7]. Unfortunately, we do not have any deeper, first-hand experiences with metaheuristic frameworks in implementation challenges, and instead refer the reader to the comparative investigation of Parejo *et al.* [11]. Note however that there is some indication that such frameworks can successfully be applied to hard optimization problems, see the results of the commercial software "LocalSolver" on the Kaggle2015 Challenge.

4 How to make the developed solution approach robust, reliable and efficient?

When contributing to implementation challenges, the aspects of robustness, reliability and efficiency of the employed method are, from our experience, most important.

"Robustness/reliability" can be interpreted such that datasets with different characteristics are solved equally well. This is of relevance as in many cases (ITC2007, NRC2010, MISTA2013, ROADEF2014, VeRoLog2014, VeRoLog2015), "hidden" instances, i. e. datasets that are not made available to the participants, are taken to verify the submitted algorithms. From our perspective, the following concepts should be given priority when trying to achieve "robustness/reliability".

- Multiple neighborhoods: the use of multiple neighborhoods, such as proposed by Variable Neighborhood Search, along with an appropriate "shaking"/perturbation, is, form our perspective, the most promising approach for a robust, reliable, and effective solution approach. We have seen this in several cases (Kaggle2012, MISTA2013, VeRoLog2014, VeRoLog2015, Kaggle2015), where virtually all successful approaches relied on the use of a set of neighborhood operators. We conclude that, when in doubt, invest in finding additional, useful neighborhoods.
- Archiving/population management [13]: While perturbation operators provide some mechanism for escaping local optima, the archiving of qualitatively good but diverse solutions should be tested, also. From our experience (VeRoLog2015), this can be a relatively easy to implement, yet effective tool.
- Robust programming: from a more software-development-point-of-view, it is recommended to double-check the obtained results from time to time. This means that additional functions should be implemented which check for constraint violations or other issues, and thus catch errors before they are overseen during search. Note that returning infeasible solutions is commonly heavily penalized (ROADEF2014).

"Efficiency" is a different animal. It has been noted before that "appropriate data structures are essential for efficient implementations of local search algorithms" [14]. From our experience, this is very true. There are numerous examples of auxiliary data structures [10] that allow fast Δ-evaluation procedures or similar. Also, the

effort spent on searching several and possibly large neighborhoods should be concentrated towards promising moves [15]. Overall, considerable development time should be allocated to achieve code efficiency. From our perspective, this is a process. It might be a good idea to first implement "naïve", i. e. slow but correct functions, and then re-implement them in an efficient way, particularly designed for the specific problem at hand (VeRoLog2015, Kaggle2015).

5 What with parameters in the model and how to tune them?

We are not aware of a "gold standard" for parameter tuning. In theory, a full factorial analysis should be conducted, but quite obviously, this can be prohibitively costly in practice. Automated configuration approaches for algorithms have been developed and successfully applied to implementation challenges ([9] to MISTA2013). Besides, algorithms can be implemented that try to self-tune the parameters during search, based on the observed search progress.

6 Conclusions

The above presented findings do not allow the formulation of general conclusions. At this points, they merely present some guidelines that might be taken into consideration for the future development of metaheuristics.

References

[1] A. Andreatta, S. Carvalho, and C. Ribeiro. *An object-oriented framework for local search metaheuristics*. In Proceedings of the 26th Conference on Technology of Object-Oriented Languages and Systems, pp. 33–45, 1998.

[2] EU/ME 2016 Workshop Website. `http://www.eume2016.be/`, last accessed February 25, 2016.

[3] A. Fink and S. Voß. *HotFrame: A Heuristic Optimization Framework*. In: S. Voß and D. L. Woodruff (eds.), Optimization Software Class Libraries Kluwer, Boston, 2002, pp. 81–154.

[4] A. Fink, S. Voß, and D. L. Woodruff. *Metaheuristic Class Libraries*. In: F. Glover and G. A. Kochenberger (eds.), Handbook of Metaheuristics, Kluwer, Boston, 2003, pp. 515–535.

[5] M. J. Geiger. *Applying the threshold accepting metaheuristic to curriculum based course timetabling*. Annals of Operations Research 194(1):189–202, 2012.

[6] W. J. Gutjahr. *Convergence Analysis of Metaheuristics*. In: V. Maniezzo, T. Stützle, and S. Voß (eds.), Matheuristics, Springer Science+Business Media, 2010, pp. 159–187.

[7] K. Helsgaun. *An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic*. European Journal of Operational Research 126(1):106–130, 2000.

[8] S. Huber and M. J. Geiger (2014). *Swap Body Vehicle Routing Problem: A Heuristic Solution Approach*. Series Lecture Notes in Computer Science 8760:16–30, 2014.

[9] M. López-Ibáñez, J. Dubois-Lacoste, T. Stützle, and M. Birattari. *The irace package, Iterated Race for Automatic Algorithm Configuration*. Technical Report TR/IRIDIA/2011-004, IRIDIA, Université libre de Bruxelles, Belgium, 2011.

[10] P. H. V. Penna, A. Subramanian, and L. S. Ochi. *An iterated local search heuristic for the heterogeneous fleet vehicle routing problem*. Journal of Heuristics 19:201–232, 2013.

[11] J. A. Parejo, A. Ruiz-Cortés, S. Lozano, and P. Fernandez. *Metaheuristic optimization frameworks: a survey and benchmarking*, Soft Computing 16:527–561, 2012.

[12] A. Schaerf, M. Lenzerini, and M. Cadoli. *LOCAL++: A C++ framework for combinatorial search problems*. Technical Report 11-99, Dipartimento di Informatica e Sistemistica, Universita di Roma La Sapienza, Rome, Italy, 1999.

[13] K. Sörensen and M. Sevaux. *MA PM: memetic algorithms with population management*. Computers & Operations Research 33(5):1214–1225, 2006.

[14] T. Stützle and I. Dumitrescu. *Topics in Local Search*. International Summer School on Metaheuristics, Playa de las Américas, Tenerife, Spain, 2003.

[15] P. Toth and D. Vigo. *The Granular Tabu Search and Its Application to the Vehicle-Routing Problem*. INFORMS Journal on Computing 15(4):333–346, 2003.

[16] C. Valouxis, C. Gogos, G. Goulas, P. Alefragis, and E. Housos. *A systematic two phase approach for the nurse rostering problem*. European Journal of Operational Research 219(2):425–433, 2012.

[17] D. L. Woodruff. *A class library for heuristic search optimization*. INFORMS Computer Science Technical Section Newsletter, 18(2):1–5, 1997.