Grid-enhanced Polylithic Modeling and Solution Approaches for Hard Optimization Problems

Josef Kallrath and Robert Blackburn and Julius Näumann

Abstract We present a grid enhancement approach (GEA) for hard mixed integer or nonlinear non-convex problems to improve and stabilize the quality of the solution if only short time is available to compute it, *e.g.*, in operative planning or scheduling problems. Branch-and-bound algorithms and polylithic modeling & solution approaches (PMSA) – tailor-made techniques to compute primal feasible points – usually involve problem-specific control parameters **p**. Depending on data instances, different choices of **p** may lead to variations in run time or solution quality. It is not possible to determine optimal settings of **p** *a priori*. The key idea of the GEA is to exploit parallelism on the application level and to run the polylithic approach on several cores of the CPU, or on a cluster of computers in parallel for different settings of **p**. Especially scheduling problems benefit strongly from the GEA, but it is also useful for computing Pareto fronts of multi-criteria problems or computing minimal convex hulls of circles and spheres. In addition to improving the quality of the solution, the GEA helps us maintain a test suite of data instances for the real world optimization problem, to improve the best solution found so far, and to calibrate the tailor-made polylithic approach.

Josef Kallrath
Department of Astronomy, University of Florida, Gainesville, FL 32611, USA, e-mail: jkallrath@ufl.edu and e-mail: josef.kallrath@web.de

Robert Blackburn
Discrete Optimization and Logistics, Karlsruhe Institute of Technology, 76133 Karlsruhe, Germany, e-mail: robert.blackburn@alumni.uni-heidelberg.de

Julius Näumann
Discrete Optimization and Logistics, Technical University of Darmstadt, 76133 Darmstadt, Germany e-mail: julius.naumann@stud.tu-darmstadt.de

1 Introduction

The term *polylithic modeling and solution approaches* (PMSA, for short) has been coined by [21] and refers to a framework or tailor-made techniques for solving hard mixed integer optimization (MIP) or non-convex nonlinear programming (ncNLP) problems exploiting several models and their solutions to establish primal feasible points, and sometimes even dual bounds. These models can be relaxations of the original MIP problem, or auxiliary models to obtain, for instance, better lower bounds on the original problem, or bounds on integer variables. The key idea of PMSA is that exact optimization algorithms and heuristics are both used to solve a MIP or ncNLP problem. Related or similar are *matheuristics* connecting mathematical programming and meta-heuristics; *cf.* [27]. Note that PMSA go beyond meta-heuristics, *i.e.*, master strategies such as evolutionary algorithms (in this group we find genetic algorithms), or local search techniques simulated annealing, or tabu search. Especially, when it comes to constrained optimization PMSA become superior.

PMSA can also establish algorithms in their own right, *e.g.*, variants of Fix-and-Relax; *cf.* Sect. 3.6.1 in [30]. Usually, such techniques involve tuning parameters $\mathbf{p}$ controlling the selection of auxiliary models or conditions under which to operate them. Depending on data instances, different choices of $\mathbf{p}$ may lead to different running times or quality of the solution. It is not possible to determine *a priori* optimal settings of $\mathbf{p}$.

To weaken the dependence of the running time and quality of the solution on $\mathbf{p}$ and to improve the quality of the solution, in this paper we propose to enhance PMSA by a multi-grid or multi-start parameter approach called *grid enhanced approach* (GEA) or *parallel PMSA* (pPMSA). The essence of this approach is to run the whole PMSA on several cores of the CPU or on a cluster of computers in parallel for a full or partial list of parameter combinations $\mathbf{p} \in \mathcal{P}$. We can either let each job for a certain parameter combination run for a certain time and extract the best solution. Alternatively, if we have a problem with a single objective function or if we are able to qualify the goodness of the solution in the multi-criteria case, we can terminate jobs if they are dominated by the current best solution. Note the difference between the multi-start parameter approach and multi-start techniques. While the latter uses multi-starts to find initial feasible points when solving ncNLP problems or when applying local search techniques by using different initial variables $\mathbf{x}$ inherent to the optimization problem, the former uses different parameters selecting algorithms, sub-models or solvers, or parameters inherent to algorithms or solvers.

Optimization is struggling with making use of today's and tomorrow's multi-core computing architecture as many of the optimization community's algorithms run inherently sequential and even for algorithms that are suitable for parallel processing (*e.g.*, branch-and-bound) the actual speed-up is limited. While usually one finds parallelization techniques deep on the level of

the solver technology, *cf.* [25], [37], [34], [7], [36] or [38], or exploiting multiple threads (*cf.* [16] or [35]) when implementing branch-and-bound based methods, our parallelization attacks at a higher level of the application itself – and is thus very problem-specific. Running the same problem with different algorithms, parameters, *etc.* and choosing the fastest one, also known as *concurrent optimization* is one way – actually the easiest one called embarrassingly parallel or perfectly parallel by [17] – to utilize the parallel computing power. GEA takes this one step further by a) applying the multi-grid approach on the level of the application itself by exploiting the control parameters of the PMSA, and b) allowing communication among the parallel runs as illustrated in the scheduling example in Section 6.2. To give a brief overview in the table below we present a few meanings of the parameters referred to by **p**:

1. choice of algorithm within a solver or PMSA,
2. control and tuning parameters of algorithms of PMSA,
3. choice of a solver,
4. control and tuning parameters of solvers.

where *solver* refers to commercial MILP solvers `CPLEX` [18], `GUROBI` [15] or `XPRESS` [16], or NLP/MINLP solvers such as `BARON` [13], `ANTIGONE` [28], or `LINDO` [26] to name a few. *Algorithm* could be, for instance, primal simplex, dual simplex or barrier in the MILP solvers. It could also be MILP or genetic algorithm for solving large traveling salesman problems. Inner parameters of solvers could be upper limits on CPU time, or numeric tolerances for satisfying constraints.

Highlights of this contribution:

1. With PMSA we present a generic framework for extending the set of hard MILP, non-convex NLP, or MINLP problems which can be solved in reasonable time.
2. The GEA on multicore platforms and clusters is generic and timeless and allows us to implement PMSA with reduced dependence on tuning and control parameters and increasing the quality of solutions if only limited time is available. Implementation issues are explained using the algebraic modeling language `GAMS`.
3. In addition to improving the quality of the solution, the GEA helps us maintain a test suite of data instances for the real world optimization problem at hand, to improve the best solution found so far, and last but not least, to calibrate the tailor-made polylithic approach.

We try not to get lost in the details of the various application examples and rather focus on the generic principles.

2 Literature Review

There are ideas and frameworks in the literature which are similar to the GEA or pPMSA but often with a different focus or motivation. Therefore, we try to cover books and articles and outline the ideas without claiming that this is complete. We identify three major areas where similar ideas or approaches are used:

1. Parallel algorithmic techniques (*concurrent*, *concurrent-distributed- concurrent*, *distributed*) within the MILP solvers `CPLEX`, `GUROBI` and `XPRESS`,
2. Parallel meta-heuristics, and
3. Machine learning and hyper-parameter optimization.

Prior to going in more depth for these three fields, we point the reader to a very useful taxonomy of parallel architectures provided by [39], pp. 522. The advantages of using multi-core platforms versus clusters of computer are discussed by [3], pp. 13.

All commercial MILP solvers allow *concurrent* runs with various flavors: *Concurrent*, *concurrent-distributed-concurrent*, *distributed*. *Concurrent* optimization for MILP can be understood as the simplest realization of the GEA and is available in `CPLEX`, `GUROBI` or `XPRESS`. The next level is *concurrent-distributed-concurrent* which allows communication and interaction between parallel runs on cores or threads. *Distributed MILP* means: Each B&B search is started with different parameter settings, a permutation of the columns/rows, or just another random seed. The best one wins, or one even allows restarts and only continues with those settings that perform best so far. `CPLEX`, for instance, offers *distributed* with the following tasks: (i) work on lower bound on one thread; (ii) work on primal bound (heuristics!) on the other, and (iii) have a third thread to manage the search tree. Impressive results are provided by [34] using a parallel enhanced version of the solver `SCIP` (*cf.* [33] or [14]) and 80,000 cores in parallel on the Titan supercomputer to solve 12 previously unsolved MILP problem from the MIPLIB benchmark set.

Although PMSA go beyond meta-heuristics, it is worthwhile to be aware of what is going on in field of parallel meta-heuristics; *cf.* [1], [4], [2], various chapters in [1] about parallel versions of genetic algorithms, simulated annealing, and tabu search, the early work by [29], [12], or [10]. If we follow [1] in his book *Parallel Metaheuristics* on p. 112, in many cases, pPMSA would fall into the class of *independent run models*.

As in Section 6.1 we provide an example in which we have used pPMSA to compute the Pareto front, we point out that there exists a vast body of literature related to parallel techniques for solving multi-objective optimization problems. This requires to construct a set of solutions called the Pareto front. [11] favor evolutionary algorithms for this. [20] construct a specially defined parallel tabu search applied to the Pareto front reached by an evolutionary algorithm.

A different community and field where parallel solution approaches have an impact is machine learning and hyper-parameter optimization in the context of Bayesian optimization. In machine learning, hyper-parameter optimization or tuning is the goal to select a set of optimal hyper-parameters for a learning algorithm. Hyper-parameters are parameters whose values are used to control the learning process, while the values of other parameters (usually node weights) are learned. Grid search and random search (*cf.* [6]) allow easy implementation to parallel approaches. [5] let a Gaussian process algorithm and a tree-structured parzen estimator run asynchronously in order to make use of multiple compute nodes and to avoid wasting time waiting for trial evaluations to complete.

What comes closest to the ideas of our paper are the possibilities presented by [9] and [16] for problem decomposition and concurrent solving from a modeling point of view with example implementations in `Mosel` that show handling of multiple models, multiple problems within a model, and as a new feature, distributed computation using a heterogeneous network of computers. In 2004 and 2012, the `XPRESS` module *mmjobs* probably focussed on solving to MILP problems or solving NLP problems with multi-start techniques. This module allows one on the modeling level to determine what to parallelize and how to distribute jobs (whole model or submodels).

3 Mathematical Structure of the Grid Approach

We want to solve a MILP, NLP, or MINLP problem $P(\mathbf{x})$ in a vector $\mathbf{x}$ of variables (continuous or integer) defined in the most general case as

$$\min f(\mathbf{x}) \quad \text{s.t.} \quad \mathbf{g}(\mathbf{x}) = 0 \wedge \mathbf{h}(\mathbf{x}) \geq 0 \quad .$$

Let us discuss first why we want to use a grid approach for solving $P(\mathbf{x})$. Reason 1: One relevant practical requirement is that we have a limit on the time available for returning a solution back to the user, *i.e.*, we usually cannot solve the problem to optimality. In this situation, we want to get the best solution within the available time. Reason 2: Problem $P(\mathbf{x})$ is a multi-criteria optimization problem, but it is hard to qualify what a *good solution* means for the owner of the problem. Therefore, we want to offer various solutions enabling the user to select by inspection the *best* solution. Note that both reasons can also show up in combination.

For both reasons, we can distinguish two cases. Case 1 ($P(\mathbf{x})$ is not too hard): We may want to solve $P(\mathbf{x})$ using different solvers, or use a specific solver with different settings of some of its tuning parameters. Case 2: If $P(\mathbf{x})$ is very hard to solve, we may want to resort to PMSA. In both case, it is not possible to determine *a priori* optimal settings of the tuning or control parameters.

Therefore, we consider variations of $P(\mathbf{x})$ named $P_{\mathbf{p}}(\mathbf{x})$ defined as

$$\min f_{\mathbf{p}}(\mathbf{x}) \quad \text{s.t.} \quad \mathbf{g}_{\mathbf{p}}(\mathbf{x}) = 0 \wedge \mathbf{g}_{\mathbf{p}}(\mathbf{x}) = 0 \quad ,$$

where $\mathbf{p} \in \mathcal{P}$ is a set of string-valued parameters specifying an instance of $P(\mathbf{x})$ or providing instructions on which solution approach or MINLP solver to use for solving $P_{\mathbf{p}}(\mathbf{x})$. Typical examples for such instructions could be related to whether to use the primal or dual Simplex algorithm when solving the MILP subproblems of $P(\mathbf{x})$, use `BARON` or `LINDO` as a deterministic global solver, problem-specific input parameters, and controlling parameters of the tailor-made polylithic modeling and solution approach.

Let n denote the number of instances to be evaluated, *i.e.*, $n = \#\mathcal{P}$. Each instance $P_{\mathbf{p}}(\mathbf{x})$ is solved on a core of the CPU or on a machine within a cluster of computers. Instances may have different run times. By k we denote the number of cores or machines. If $n \leq k$, we can solve all instances in parallel. In case $n > k$, we have up to k instances running in parallel. If instance $P_{\mathbf{p}}(\mathbf{x})$ has finished, we submit the next instance $P_{\mathbf{p+1}}(\mathbf{x})$ to be solved. The result of $P_{\mathbf{p}}(\mathbf{x})$ may enable us to terminate an actively running instance $P_{\tilde{\mathbf{p}}}(\mathbf{x})$.

4 Optimization Problems and Optimization Algorithms Suitable for the Grid Approach

Structurally, there are two categories of optimization problems which benefit from GEA: Problems where it is difficult to find a good feasible point in short time, or multi-criteria optimization problems. An example for a multi-criteria optimization problem is the simultaneous minimization of trimloss and the number of patterns in 1D cutting stock problems as described and solved in [24], hereafter referred to as PCSP. Scheduling problems in the process industry (*cf.* [19], [32], or [8]) or lock scheduling problems (*cf.* [40]) are solved exploiting polylithic techniques – they are also multi-criteria in nature. Lock scheduling problems (*cf.* [40]) are also very suitable for this. The convex hull minimization problems treated in [23] and [22] have also been solved by polylithic approaches using various homotopy techniques in which a preliminary model is exploited to generate a feasible starting point which is then improved. The stronger the sensitivity of a problem *w.r.t.* some tuning parameters, the more suitable and efficient the GEA becomes to enhance the PMSA.

In addition to the inherent structure of an optimization problem regarding the suitability of GEA, the algorithms themselves used to solve a difficult optimization problem can also make the usage of a GEA attractive. Very suitable are meta-heuristics, *e.g.*, genetic algorithms. Hyper-parameter grid search in machine learning had already been mentioned in Section 2. Parallel

search in constraint programming is also a suitable technique for GEA; *cf.* [31].

5 IT-Aspects and Implementation

5.1 Generic Structure

As in Figure 1, the grid approach can be structured into the following modules (larger pieces or collections of several pieces of programming code):

1. Model M_0 corresponding to $P(\mathbf{x})$.
2. Module M_v creating variations V_n of model M_0, *e.g.*, relaxations of $P(\mathbf{x})$, or related, auxiliary models of $P(\mathbf{x})$, combined with variations of solver configuration.
3. Module M_g defining and generating the instances
4. Module M_c submitting, controlling, evaluating and possibly terminating optimization runs of the instances
5. Module M_r collecting the results

5.2 Implementation in `GAMS`

We have implemented the GEA in `GAMS`, but in principle, it could be implemented in any algebraic modeling or programming language. Here we provide an explanation involving some `GAMS` flavor. The `GAMS` program *application.gms* is the main `GAMS` file to be executed. It contains the monolithic model and the PMSA controlled by *cntrl.txt* or *compile.par* containing various scalars and compile-time parameters. It calls *multi-start.gms*, which triggers M_v to generate variations. Regarding our `GAMS` implementation, the varied instance parameters can be either string-valued compile-time parameters or numerical scalars. *Multi-start.gms* in turn calls *application.gms* asynchronously for each variation to create an instance and run it (module M_g). The maximum number of cores to be used can be configured through a parameter in *cntrl.txt* or *compile.par* read by *application.gms*. All runs are administered (module M_c) in *multi-start.gms*, which also collects the final results (module M_r).

5.2.1 Module M_v

There are two ways to generate and handle the variations, either automatically or manually: Each generated variation will be assigned a number and is stored in a *.gmi* file named accordingly if generated *a priori* by a Python

script. This approach is preferred if we want to generate all combinations of tuning and control parameters. The Python script creates variations based on input from a file. The following is an example of such a file, containing instructions to vary the solver to be used and a scalar parameter:

```
SOLVER , C, L {\QTR{tt}{GUROBI},CBC}
Param_A, S, L {0.250,0.500}
```

Alternatively, variations can also be supplied directly as stored or programmed, respectively, in *application_ParSet.gms*. This approach has advantages when it is not possible to generate all combinations of parameters, *e.g.*, when dealing with solver parameters not available in all solvers.

5.3 Module M_g

Module M_g will prepare each variation for being run with `GAMS` by creating subdirectories containing the according .gmi files.

5.3.1 Modules M_c and M_r

Modules M_c and M_r are both implemented in *multi-start.gms*. M_c asynchronously submits each run and constantly watches for results. If a sufficient solution has been returned by a run, it will terminate the others. Module M_r has the function of specifying the best solution. For single-objective function problems, the best solution is obvious. For multi-criteria problems we need to proceed differently. We define and construct a solution metric which allows us to decide automatically which solution is the best.

6 Real World Examples

6.1 Cutting Stock Pareto Front

This problem consists of the simultaneous minimization of trim loss and patterns in cutting stock problems (CSPs). The problem is solved by a PMSA described in [24] which is essentially an *exhaustion approach* combining a greedy approach (maximize pattern multiplicity) with a MILP formulation of the CSP containing various tuning parameters among them $w_{\max}$, the maximal permissible waste per pattern as the most important control parameter. Based on the GEA, the Pareto front is computed as a function of $w_{\max}$. In this simple case, the GEA only exploits parallelism by computing the Pareto front simultaneously for six different values of $w_{\max}$: 20, 15, 10, 8, 6 and 4%,

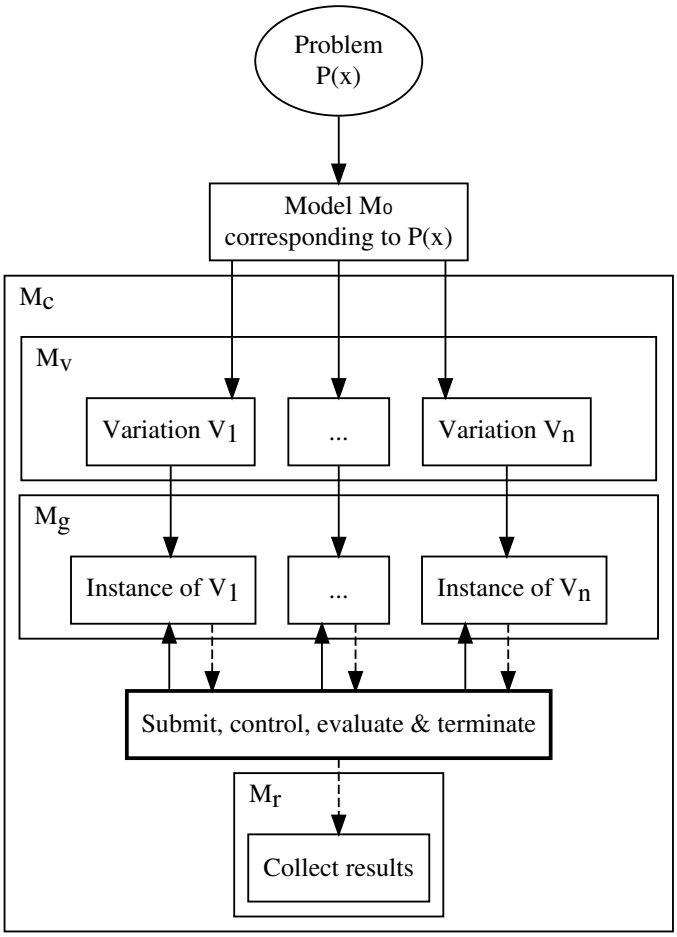

Fig. 1 The modules of the grid approach. Module M_c is the master module with over-all control. Variations V_n are generated by M_v, possibly *a priori* by a Python script, and run by M_g. Results are collected by M_r reading from the directories generated by M_g.

which results in a computational speedup of approximately a factor 6 as the jobs are independent. If the Pareto front of any multi-criteria optimization problem can be generated as a function of one or several parameters, it can be generated exploiting the GEA.

6.2 Scheduling Problem in the Process Industry

Consider the plant system with many reactors, tanks, continuous units *etc.* leading to a multi-criteria scheduling problem solved by [19]. The core of the problem is a MILP model exploiting a continuous time formulation involving event points. It is solved polylithically by a *Moving time window* approach, in which the tunable parameters are maximum delay, maximum underproduction, and the maximum number of event points, as well as penalty coefficients on time and production target deviations.

In the GEA we combine the tuning parameters of the approach and distinguish solution metrics for

1. make-to-order (bulk articles, mtoB),
2. make-to-order (packed articles, mtoP),
3. make-to-stock articles (mts), and
4. all products.

Within the GEA we trace the following criteria c:

c	*description*
01-04	relative underproduction (mtoB, mtoP, mts, all)
05-08	relative overproduction in %
09-12	delays
13-16	earliness
17	number of changeovers
18-21	ratio of priority 0/1/2/3 over all tasks
22	elapsed time (more for just knowing it)

Jobs run for 30 minutes at most. If the deviations from time and production targets, or a combined metric of them, become sufficiently small for a job, the solution is considered good and all other jobs not yet finished are terminated.

An explorative test performed on a cluster ran with up to 1,000 jobs providing good schedules. In practical operative situations with running time constraints, one should not create more parameter combinations than cores or computing units available.

6.3 2D Minimal Perimeter Convex Hulls

Given a set of n circles with radii R_i and a rectangle with length L and width W. Find a configuration of non-overlapping circles (specified by their center coordinates) which fit into the rectangle and do not overlap and lead to a convex hull whose perimeter has minimal length. The monolithic model M has been developed by [22]; it is a MINLP problem with bilinear, 4th-order polynomial, and trigonometric terms. The PMSA consists of different

approaches to solve the MINLP problem with simplified models providing initial starting values for M:

1. P1: Minimize area or perimeter of a rectangle hosting the circles to produce initial values for M.
2. P2: Minimize the weighted distances of circles to center of circles to produce initial values for M.
3. T: Tour-specified approach.

The GEA consists of parallel runs over M, P1, P2, and T. In this example, the GEA is helpful in developing efficient numeric schemes and in exploiting the current hardware as well as possible.

6.4 3D Minimal Surface Area Convex Hulls

Given a set of n spheres with radii R_i. Find a configuration of non-overlapping spheres (specified by their center coordinates) which lead to a convex hull whose boundary has minimal area. The monolith model M, dominated by bilinear terms, has been developed by [22]. The PMSA consists of various approaches to solve the NLP problem with simplified models providing initial starting values for M:

1. P1: Minimize the volume of a sphere or a rectangular box hosting the spheres to produce initial values for M.
2. P2: Minimize the weighted distances of spheres to the center of spheres to produce initial values for M.

The GEA consists of parallel runs over M, P1, P2. As above, the GEA is helpful in developing efficient numeric schemes and in exploiting the current hardware as well as possible.

7 Test Suite of Data Instances

In our development work, the GEA helps us to maintain a test suite of data instances for the real world optimization problem at hand, to improve the best solution found so far, and last but not least, to calibrate the tailor-made polylithic approach. The test suite is populated by real word data instances. Accumulated over several month and years, the collected data instances represent real world situations more and more appropriately. Test runs over this test suite take longer and longer. Thus, the GEA is of great help in covering as many parameter combinations as possible. Eventually, we learn that certain parameter combinations are dominated by others. Especially for scheduling problems, it is usually computationally prohibitive to compute the

strict global optimum. Therefore, at best, we find a good feasible solution. Script files automatically evaluate the results of the individual data instances involved in the test runs, detect whether we have obtained a solution better than the previous best solution, and thus improve our test suite.

8 Conclusions and Discussions

We have constructed a generic grid enhancement approach (GEA) in connecting with polylithic modeling and solution approaches (PMSA) to solve a hard mixed integer or non-convex nonlinear optimization problem when a solution has to be returned within a given time limit, or when the user wants to inspect various solutions, for instance, in a multi-criteria optimization problem. The GEA works on one computer with several cores or on a cluster of computers. On each core or cluster computer we solve the problem at hand with possibly different solvers, different solver parameters, different algorithms in the sense of PMSA or different tuning parameters of the algorithms at hand. The approach is very useful for operative planning or scheduling problems.

Although the idea of the GEA is simple in nature, great care has to be paid to implementation in whatever programming language regarding robustness and maintainability of the code. A valid point of discussion is the notice that PMSA increases the complexity of implemented decisions systems, and thus, also the maintenance effort. The crucial question to be answered is: Should everything which is possible also be done in mathematical optimization? The answer depends on the importance and the value of the decision problem. We have just tried to keep the approach as generic and clean as possible. Implementing the PMSA may take weeks or a month. The implementation for enhancing PMSA using the grid approach requires rather days, weeks, or possibly a month, including testing.

However, the GEA provides another important advantage. For real word optimization problems, testing is very important for robustness. Therefore, we usually develop a test suite in which various problem instances with their optimal or best found solutions are stored. Grid enhanced PMSA allow us to improve a test suite, and, if dominant parameter combinations can be identified, to calibrate the tailor-made PMSA.

For the future, there is room for improvement: If it is possible to specify *a priori* what is a good solution acceptable for the user, we have a weak interaction between parallel jobs in the sense of jobs already being dominated by the one just yielding an acceptable solution. The interaction could be deepened by evaluation pre-processing models in parallel, extracting and considering the information obtained from them, and exploiting this in follow-up computations – also in parallel.

Acknowledgement

The authors are indebted to the anonymous referees whose comments helped to improve this paper. We thank Dr. Michael Bussieck (GAMS GmbH, Frechen, Germany) for discussion on parallelism used in optimization, Dr. Jens Schulz & Dr. Susanne Heipcke (FICO, Berlin, Germany & Marseille, France) for hints and details on parallelization in `XPRESS`, and Prof. Dr. Michael Torsten Koch (ZIB Berlin, Berlin, Germany), Dr. Jens Schulz and Dr. Steffen Klosterhalfen (Mannheim, Germany) for their careful reading of and feedback on the manuscript.

References

1. Alba, E.: Parallel Metaheuristics: A New Class of Algorithms. Wiley-Interscience, New York, NY, USA (2005)
2. Alba, E., Luque, G.: In: E. Alba (ed.) Parallel Metaheuristics: A New Class of Algorithms, Wiley Series on Parallel and Distributed Computing, chap. 2. Measuring the Performance of Parallel Metaheuristics, pp. 43–62. Wiley (2005)
3. Alba, E., Luque, G., Nesmachnow, S.: Parallel Metaheuristics: Recent Advances and New Trends. ITOR **20**(1), 1–48 (2013)
4. Alba, E., Talbi, E.G., Luque, G., Melab, N.: In: E. Alba (ed.) Parallel Metaheuristics: A New Class of Algorithms, Wiley Series on Parallel and Distributed Computing, chap. 4. Metaheuristics and Parallelism, pp. 79–104. Wiley (2005)
5. Bergstra, J., Bardenet, R., Bengio, Y., Kégl, B.: Algorithms for Hyper-parameter Optimization. In: Proceedings of the 24th International Conference on Neural Information Processing Systems, NIPS'11, pp. 2546–2554. Curran Associates Inc., USA (2011)
6. Bergstra, J., Bengio, Y.: Random Search for Hyper-Parameter Optimization. J. Mach. Learn. Res. **13**, 281–305 (2012)
7. Berthold, T., Farmer, J., Heinz, S., Perregaard, M.: Parallelization of the FICO xpress-optimizer. Optimization Methods and Software **33**(3), 518–529 (2018)
8. Borisovsky, P.A., Eremeev, A.V., Kallrath, J.: Reducing the Number of Changeover Constraints in a MIP Formulation of a Continuous-Time Scheduling Problem. arXiv e-prints arXiv:1408.5832 (2014)
9. Colombani, Y., Heipcke, S.: Multiple Models and Parallel Solving with Mosel. Tech. rep., FICO Xpress Optimization, Birmingham, UK. URL `http://www.fico.com/fico-xpress-optimization/docs/latest/mosel/mosel{_}parallel/dhtml`
10. Crainic, T.G.: Parallel metaheuristics and cooperative search. In: M. Gendreau, J.Y. Potvin (eds.) Handbook of Metaheuristics, pp. 419–451. Springer (2019)
11. Figueira, J., Liefooghe, A., Talbi, E.G., Wierzbicki, A.: A Parallel Multiple Reference Point Approach for Multi-objective Optimization. European Journal of Operational Research **205**(2), 390 – 400 (2010). DOI https://doi.org/10.1016/j.ejor.2009.12.027. URL `http://www.sciencedirect.com/science/article/pii/S0377221710000081`
12. Gendreau, M., Potvin, J.Y.: Handbook of Metaheuristics, 2nd edn. Springer Publishing Company, Incorporated (2010)
13. Ghildyal, V., Sahinidis, N.V.: Solving Global Optimization Problems with BARON. In: A. Migdalas, P. Pardalos, P. Varbrand (eds.) From Local to Global Optimization. A Workshop on the Occasion of the 70th Birthday of Professor Hoang Tuy, chap. 10, pp. 205–230. Kluwer Academic Publishers, Boston, MA (2001)

14. Gleixner, A., Bastubbe, M., Eifler, L., Gally, T., Gamrath, G., Gottwald, R.L., Hendel, G., Hojny, C., Koch, T., Lübbecke, M.E., Maher, S.J., Miltenberger, M., Müller, B., Pfetsch, M.E., Puchert, C., Rehfeldt, D., Schlösser, F., Schubert, C., Serrano, F., Shinano, Y., Viernickel, J.M., Walter, M., Wegscheider, F., Witt, J.T., Witzig, J.: The SCIP Optimization Suite 6.0. Technical report, Optimization Online (2018). URL http://www.optimization-online.org/DB_HTML/2018/07/6692.html
15. Gurobi Optimization, L.: Gurobi Optimizer Reference Manual (2019). URL http://www.gurobi.com
16. Heipcke, S.: Xpress-Mosel: Multi-Solver, Multi-Problem, Multi-Model, Multi-Node Modeling and Problem Solving. In: J. Kallrath (ed.) Algebraic Modeling Systems: Modeling and Solving Real World Optimization Problems, pp. 77–110. Springer, Heidelberg, Germany (2012)
17. Herlihy, M., Shavit, N.: The Art of Multiprocessor Programming, Revised Reprint, 1st edn. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA (2012)
18. IBM: IBM ILOG CPLEX Optimization Studio (2017) CPLEX Users Manual (2017). URL http://www.ibm.com
19. Janak, S.L., Floudas, C.A., Kallrath, J., Vormbrock, N.: Production Scheduling of a Large-Scale Industrial Batch Plant: I. Short-Term and Medium-Term Scheduling. Industrial and Engineering Chemistry Research **45**, 8234–8252 (2006a)
20. Jozefowiez, N., Semet, F., Talbi, E.G.: Parallel and Hybrid Models for Multi-objective Optimization: Application to the Vehicle Routing Problem. In: J.J.M. Guervós, P. Adamidis, H.G. Beyer, H.P. Schwefel, J.L. Fernández-Villacañas (eds.) Parallel Problem Solving from Nature — PPSN VII, pp. 271–280. Springer Berlin Heidelberg, Berlin, Heidelberg (2002)
21. Kallrath, J.: Polylithic Modeling and Solution Approaches Using Algebraic Modeling Systems. Optimization Letters **5**, 453–466 (2011). 10.1007/s11590-011-0320-4
22. Kallrath, J., Frey, M.M.: Minimal Surface Convex Hulls of Spheres. Vietnam Journal of Mathematics **46**, 883–913 (2018)
23. Kallrath, J., Frey, M.M.: Packing Circles into Perimeter-Minimizing Convex Hulls. Journal of Global Optimization **73**(4), 723–759 (2019). DOI https://doi.org/10.1007/s10898-018-0724-0
24. Kallrath, J., Rebennack, S., Kallrath, J., Kusche, R.: Solving Real-World Cutting Stock-Problems in the Paper Industry: Mathematical Approaches, Experience and Challenges. European Journal of Operational Research **238**, 374–389 (2014)
25. Laundy, R.S.: Implementation of Parallel Branch-and-Bound Algorithms in Xpress-MP. In: T.A. Ciriani, S. Gliozzi, E.L. Johnson, R. Tadei (eds.) Operational Research in Industry. MacMillan, London (1999)
26. Lindo Systems: Lindo API: User's Manual. Lindo Systems, Inc., Chicago (2004)
27. Maniezzo, V., Sttzle, T., Vo, S.: Matheuristics: Hybridizing Metaheuristics and Mathematical Programming, 1st edn. Springer Publishing Company, Incorporated (2009)
28. Misener, R., Floudas, C.: ANTIGONE: Algorithms for coNTinuous / Integer Global Optimization of Nonlinear Equations. Journal of Global Optimization **59**, 503–526 (2014). DOI 10.1007/s10898-014-0166-2
29. Pardalos, P.M., Pitsoulis, L.S., Mavridou, T.D., Resende, M.G.C.: Parallel Search for Combinatorial Optimization: Genetic Algorithms, Simulated Annealing, Tabu Search and GRASP. In: Parallel Algorithms for Irregularly Structured Problems, Second International Workshop, IRREGULAR '95, Lyon, France, September 4-6, 1995, Proceedings, pp. 317–331 (1995). DOI 10.1007/3-540-60321-2_26. URL https://doi.org/10.1007/3-540-60321-2_26
30. Pochet, Y., Wolsey, L.A.: Production Planning by Mixed Integer Programming. Springer, New York (2006)
31. Régin, J.C., Malapert, A.: Parallel Constraint Programming. In: Y. Hamadi, L. Sais (eds.) Handbook of Parallel Constraint Reasoning, pp. 337–379. Springer International Publishing (2018)

32. Shaik, M.A., Floudas, C.A., Kallrath, J., Pitz, H.J.: Production Scheduling of a Large-Scale Industrial Continuous Plant: Short-Term and Medium-Term Scheduling. Computers and Chemical Engineering **33**, 670–686 (2009)
33. Shinano, Y., Achterberg, T., Berthold, T., Heinz, S., Koch, T.: ParaSCIP: A Parallel Extension of SCIP. In: Competence in High Performance Computing 2010 - Proceedings of an International Conference on Competence in High Performance Computing, Schloss Schwetzingen, Germany, June 2010., pp. 135–148 (2010)
34. Shinano, Y., Achterberg, T., Berthold, T., Heinz, S., Koch, T., Winkler, M.: Solving Open MIP Instances with ParaSCIP on Supercomputers Using up to 80,000 Cores. In: 2016 IEEE International Parallel and Distributed Processing Symposium (IPDPS), pp. 770–779 (2016)
35. Shinano, Y., Berthold, T., Heinz, S.: A First Implementation of ParaXpress: Combining Internal and External Parallelization to solve MIPs on Supercomputers. In: International Congress on Mathematical Software, pp. 308–316. Springer (2016)
36. Shinano, Y., Berthold, T., Heinz, S.: ParaXpress: An Experimental Extension of the FICO Xpress-Optimizer to Solve Hard MIPs on Supercomputers. Optimization Methods & Software (2018). DOI 10.1080/10556788.2018.1428602. Accepted for publication on 2018-01-13
37. Shinano, Y., Fujie, T., Kounoike, Y.: Effectiveness of Parallelizing the ILOG-CPLEX Mixed Integer Optimizer in the PUBB2 Framework. In: K. H., L. Böszörményi, H. Hellwagner (eds.) Euro-Par 2003 Parallel Processing. Euro-Par 2003, *Lecture Notes in Computer Science*, vol. 2790, pp. 770–779 (2003). DOI 10.1109/IPDPS.2016.56
38. Shinano, Y., Heinz, S., Vigerske, S., Winkler, M.: FiberSCIP - A Shared Memory Parallelization of SCIP. INFORMS Journal on Computing **30**(1), 11 – 30 (2018). DOI 10.1287/ijoc.2017.0762
39. Trelles, O., Rodriguez, A.: In: E. Alba (ed.) Parallel Metaheuristics: A New Class of Algorithms, Wiley Series on Parallel and Distributed Computing, chap. 21. Bioinformatics and Parallel Metaheuristics, pp. 517–549. Wiley (2005)
40. Verstichel, J., De Causmaecker, P., Spieksma, F., Vanden Berghe, G.: Exact and Heuristic Methods for Placing Ships in Locks. European Journal of Operational Research **235**(2), 387–398 (2014). DOI 10.1016/j.ejor.2013.06.045. URL `https://lirias.kuleuven.be/handle/123456789/403645`