DE GRUYTER OPEN

JAISCR, 2017, Vol. 7, No. 3, pp. 171 – 181
10.1515/jaiscr-2017-0012

MIDACO PARALLELIZATION SCALABILITY ON 200 MINLP BENCHMARKS

Martin Schlueter and Masaharu Munetomo

Information Initiative Center, Hokkaido University Sapporo, Sapporo 060-0811, Japan

Submitted: 16th September 2016; accepted: 15th November 2016

Abstract

This contribution presents a numerical evaluation of the impact of parallelization on the performance of an evolutionary algorithm for mixed-integer nonlinear programming (MINLP). On a set of 200 MINLP benchmarks the performance of the MIDACO solver is assessed with gradually increasing parallelization factor from one to three hundred. The results demonstrate that the efficiency of the algorithm can be significantly improved by parallelized function evaluation. Furthermore, the results indicate that the scale-up behaviour on the efficiency resembles a linear nature, which implies that this approach will even be promising for very large parallelization factors. The presented research is especially relevant to CPU-time consuming real-world applications, where only a low number of serial processed function evaluation can be calculated in reasonable time.

Keywords: MINLP, optimization, MIDACO, parallelization

1 Introduction

This contribution is an extended version of the numerical study recently presented in Schlueter and Munetomo [23]. This study discusses the optimization of problems known as mixed-integer nonlinear programs (MINLP). The considered MINLP is stated mathematically in (1), where $f(x,y)$ denotes the objective function to be minimized. In (1), the equality constraints are given by $g_{1,\ldots,me}(x,y)$ and the inequality constraints are given by $g_{me+1,\ldots,m}(x,y)$. The solution vector x contains the continuous decision variables and the solution vector y contains the discrete decision variables (also called *integers*). Furthermore, some box constraints as x_l, y_l (lower bounds) and x_u, y_u (upper bounds) for the decision variables x and y are considered in (1).

$$\begin{aligned} &\text{Minimize} \quad f(x,y) \qquad (x \in \mathbb{R},\ y \in \mathbb{Z}), \\ &\text{subject to: } g_i(x,y) = 0,\ i = 1,\ldots,m_e \\ &\qquad\qquad g_i(x,y) \geq 0,\ i = m_e+1,\ldots,m, \\ &\text{and bounds: } x_l \leq x \leq x_u\ (x_l,\ x_u \in \mathbb{R}) \\ &\qquad\qquad y_l \leq y \leq y_u\ (y_l,\ y_u \in \mathbb{N}). \end{aligned} \tag{1}$$

Optimization of MINLP problems is a young and growing field in the evolutionary computing community, see e.g. Babu and Angira [1], Cardoso [2], Costa and Oliveira [3], Deep et al. [4], Glover [6], Liang et al. [11], Mohammed [12], Munawar [13], Wasanapradit et al. [28], Young et al. [30], Yiqing et al. [29] or Yue et al. [32]. One advantage of evolutionary algorithms is their robustness towards the analytical properties of the objective and constraint functions. Therefore, in above MINLP

(1) the functions $f(x,y)$ and $g(x,y)$ are considered as *general black-box functions* without any requirements, such as differentiability or smoothness. Another advantage of evolutionary algorithms is their capability to (greatly) benefit from parallelization, see for example Du et al. [14], Laessig and Sudholt [10], Gupta [10], Sakuray [16], Sudholt [27] or Yingyong et al [31]. One of the most popular strategies to use parallelization in evolutionary algorithms is the distributed computing of the problem function evaluations. This strategy is sometimes denoted as *co-evaluation*.

Presented here numerical study investigates the impact of a varying co-evaluation factor on the performance of an evolutionary optimization algorithm on a set of 200 MINLP benchmarks (see Schittkowski [24]), which mostly originate from the well-known GAMS MINLPlib library [9]. Considered benchmark instances consist of up to 205 variables and 283 constraints, including up to 100 equality constraints (see the Appendix for details on the benchmark instances). This paper focuses on investigation and measurement of the efficiency of parallelized function evaluation calls on the algorithmic performance. As numerical solver, the MIDACO optimization software is used, which is based on an evolutionary algorithm especially developed for mixed-integer problems and capable of seamless parallelization of function evaluation calls.

This paper is structured as follows: In Section 2 a brief overview on the MIDACO algorithm is given with an emphasis on its parallelization approach. In Section 3 the numerical results of 300 individual test runs on the set of 200 MINLP benchmarks are illustrated and discussed. In Section 4 a summary and general conclusions are presented. A comprehensive Appendix lists detailed individual information on all considered benchmarks.

2 MIDACO Algorithm

MIDACO stands for *Mixed Integer Distributed Ant Colony Optimization*. The evolutionary algorithm within MIDACO is based on the ant colony optimization metaheuristic for continuous search domains proposed by Socha and Dorigo [26] and was extended to mixed-integer domains by Schlueter et al. in [17]. For constrained optimization problems the algorithm applies the *Oracle Penalty Method* which was introduced in Schlueter and Gerdts [18]. While the MIDACO algorithm is conceptually designed as general black-box solver, it has proven its effectiveness especially on challenging interplanetary space trajectory design problems (see Schlueter [21]), where it holds several best known record solutions on benchmarks provided by the European Space Agency [5]. It is furthermore the first algorithm that was able to successfully solve interplanetary trajectory problems formulated as mixed-integer problems, where the sequence of fly-by planets was considered as integer optimization variables (see Schlueter et. al. [20]).

2.1 Parallelization Approach

The parallelization approach considered here aims on distributing the problem function evaluation. This approach is sometimes referred to as co-evaluation. The MIDACO solver optimization software easily enables this kind of parallelization due to its *reverse communication* architecture. *Reverse communication* means here that the call of the objective and constraint functions happens outside and independently of the MIDACO source code.

Within a single reverse communication loop, MIDACO does accept and returns an arbitrary large number of **P** iterates x (also called "solution candidates" or "individuals") at once. Hence, those **P** iterates can be evaluated in parallel, outside and independently from the MIDACO source code. This idea of passing a **block** of **P** iterates at once within one reverse communication step to the optimization algorithm was originally introduced by the code NLPQLP by Schittkowski [25].

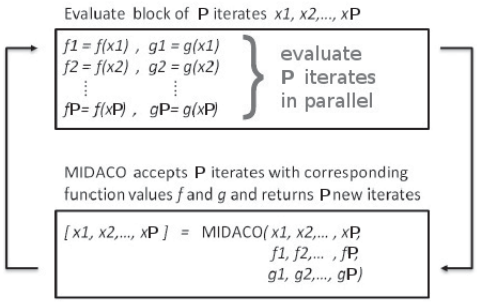

Figure 1. Reverse communication loop with block of **P** iterates (solution candidates).

Figure 1 illustrates the reverse communication loop where a **block** of **P** iterates is evaluated regard-

ing their objective function $f(x)$ and constraints $g(x)$ and then passed to the MIDACO optimization algorithm, which then again returns a new block of **P** iterates to be evaluated.

This concept allows an independent and user controlled distributed computing of the objective and constraint function. In other words: The displayed parallelization option is valid for any language and any CPU architecture. This includes in particular multi-core PC's, PC-Clusters and GPGPU (General Purpose Graphical Processing Unit) based computation. In case of MIDACO, the parallelization factor **P** can furthermore be any arbitrary large integer value, enabling a seamless and massive parallelization. As this parallelization approach aims on distributing the function evaluation calls, it is intended for problems where the function evaluation are numerically expensive to compute, which is often the case for complex real-world applications. For further details on the parallelization approach by MIDACO, please consult [22] or [19].

3 Numerical Results

This Section presents the numerical results obtained by MIDACO (5.0 beta version) on the set of 200 MINLP benchmark problems, provided by Schittkowski [24]. In total, 300 executions on the full set of 200 problem instances have been conducted. Each execution considered a different parallelization factor **P** (see Section 2.1) and a different random seed. With each execution, the parallelization factor **P** was incrementally increased from one up to three hundred. For each individual problem out of the library a maximal number of function evaluation budget of ten million was assigned. No time limit was enforced on any run. Each individual test run on each problem was either stopped if the maximal evaluation budget was reached, or if the global[1] optimal solution was obtained.

The criteria for reaching a global optimal solution (x^*, y^*) by an approximation $(\hat{x}, \hat{y})$ is given in Equation (2).

$$\begin{aligned} f(\hat{x},\hat{y}) &\leq f(x^*,y^*) + \frac{\|f(x^*,y^*)\|}{100}, \\ \|g(\hat{x},\hat{y})_{i=1,\ldots,m_e}\| &\leq 0.01, \\ g(\hat{x},\hat{y})_{i=m_{e+1},\ldots,m} &\geq -0.01. \end{aligned} \tag{2}$$

Equation (2) implies that a test run was considered successful, if the approximative solution $(\hat{x}, \hat{y})$ reached by MIDACO was as close as 1% to the global optimal solution objective function value $f(x^*, y^*)$ while satisfying all constraints with a precision of at least 0.01. Note that the tolerance of 0.01 for the constraint violation is chosen here rather moderate. This is due to the relatively large number of (up to one hundred) equality constraints in several benchmark instances (see Appendix). For real-world problems, solutions with higher precision in the constraint satisfaction can normally be achieved easily with refinement runs. The lower bounds of each problem instance were used as starting point and the original bounds[2] provided by Schittkowski [24] were considered for each problem. Except for the parallelization factor, all MIDACO parameters were set to default.

Table 1 displays the number of optimal solutions obtained for various test runs on the full set of 200 MINLP benchmarks. The number of each run equals the parallelization factor **P** used in such run. The abbreviations for Table 1 are as follows:

Run = **P**	:	Individual run on 200 benchmarks (using a parallelization factor of **P**)
Optimal	:	Number of global optimal solutions
Blocks	:	Average performed blocks
Evaluation	:	Average number of evaluation

All numerical runs were conducted on a Desktop computer with XEON cpu with 3.47GHz clockrate, 4GB RAM memory and six physical cores. The total time to calculate all 300 executions on the full benchmark library took 521,009 seconds, which is around six days. It is important to note

[1] The best known numerical $f(x,y)$ values provided in Schittkowski [24] were used as global optimal solutions throughout this study.

[2] Note that the original bounds provided in Schittkowski [24] on the problem instance are sometimes huge in the context of evolutionary computing, where the entire search space is sampled. This makes some of the instance exceptionally hard to solve with evolutionary methods.

that for the here presented results the co-evaluation of objectives and constraints was calculated on a *single thread* and not distributed by common parallelization schemes, such as OpenMP [8] or MPI [8]. The reason is that all benchmark function in this study are computational *cheap* (take only milliseconds to compute) and any actual parallelized computing scheme would introduce a computing overhead which would in fact increase overall calculation times rather than reducing them. The presented results nevertheless accurately represent the factor of reduced serial processed function evaluation and that the results are therefore fully valid to estimate the performance gain in parallel executed function evaluation for CPU-time intensive real-world applications.

Table 1. Number of optimal solutions obtained by MIDACO in various runs on 200 Benchmarks

Run = **P**	Optimal	Blocks	Evaluation
1	160	2,747,356	2,747,356
10	160	284,719	2,847,195
20	156	162,785	3,255,716
30	155	107,293	3,218,808
40	159	82,824	3,312,999
50	156	67,314	3,365,719
100	152	38,214	3,821,471
150	151	27,099	4,064,936
200	144	22,689	4,537,802
250	148	17,980	4,495,111
300	144	15,979	4,793,737

From Table 1 it can be seen that MIDACO obtained in its first run (which had a parallelization factor of one and therefore no actual parallelization) a number of 160 global optimal solutions on the set of 200 benchmarks. It is important to note that the number of processed **blocks** in the first run equals the number of evaluation, which where about 2.7 million (2,747,356). Table 1 illustrates that the number of global solutions obtained by MIDACO in regarding to various individual test runs from one to three hundred. Note in Table 1 that for an increased parallelization factor **P** the average number of blocks decreases while the average number of total performed function evaluation increases. While in the first run around 2.7 million blocks had to be processed on average, it is only a 15,979 blocks for parallelization factor of 300. Therefore a reduction of around $\frac{2747356}{15979} \approx 171.9$ times in the number of processed blocks could be achieved for the maximalconsidered parallelization factor of **P**=300, while the number of optimal solved instances dropped only by 10.0% (144 in comparison to 160).

Additionally to Table 1, the numerical results are illustrated in Figure 2 which displays the individual and average[1] number of optimal optimal solutions obtained by MIDACO in each of the 300 executions on the full benchmark library.

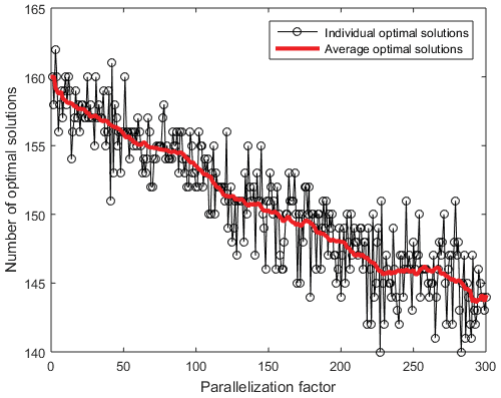

Figure 2. **Optimal** solutions at run 1 to 300.

From Table 1 it can be seen how the MIDACO algorithm benefits in drastically reducing its number of processed blocks by parallelization, while still maintaining a similar high number of global optimal solutions. From Figure 2 it can be seen that such trend exhibits a more or less linear behaviour.

In order to give a more sophisticated answer to the question, how the efficiency of the algorithm scales with the parallelization factor, a new criteria for the algorithmic efficiency is now introduced here. Based on the first run, which exemplifies the unparalleled performance, the "*Efficiency*" of a run (which number equals its parallelization factor) should be measured as given in Equation (3)

$$Efficiency(run) = \frac{\#Optimal(run)}{\#Optimal(1^{st}\ run)} \cdot \frac{\#Blocks(1^{st}\ run)}{\#Blocks(run)}. \quad (3)$$

[1]The *smooth* function provided by Matlab was used to approximate the average from the raw data of individual global optimal solutions.

Equation (3) measures the "*Efficiency*" based on a multiplication of a ratio of optimal obtained solutions with a ratio of required blocks. Because the number of optimal solutions is desired to be as high as possible, the average number of optimal solutions appear in the numerator of the ratio, while the number of optimal solutions from the first run appear in the denominator of the ratio. Contrary to desired number of optimal solutions, the number of blocks is desired to be as low as possible and hence the blocks required in the first run appear in the numerator, while the average number of blocks appear in the denominator. The efficiency measure given by Equation 3 can be calculated for each of the 300 runs on the full library of test problems. Figure 3 displays the efficiency measure for all such 300 runs.

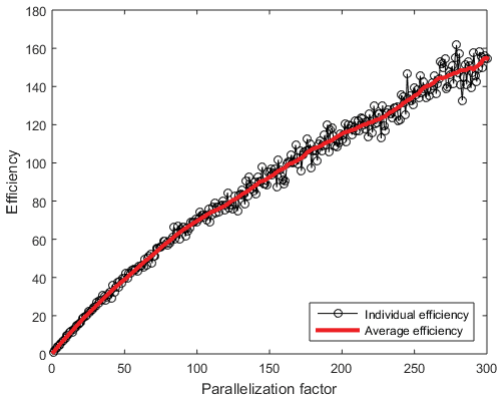

Figure 3. Dependence of efficiency on parallelization

Table 2. Efficiency of **P** factor

Run = **P**	$\frac{\#Blocks(1^{st}\ run)}{\#Blocks(run)}$	$Efficiency(\mathbf{P})$
1	1.0	1.0
10	9.6	9.5
20	16.8	16.6
30	25.6	27.2
40	33.1	32.4
50	40.8	39.7
100	71.8	68.9
150	101.3	95.2
200	121.0	112.0
250	152.8	139.3
300	171.9	154.7

Table 2 lists some of the individual efficiency measure values for various runs and additionally displays the second term ($\frac{\#Blocks(1^{st}\ run)}{\#Blocks(run)}$) of the efficiency measure formula given in Equation (3).

From Figure 3 it can be observed that the scale up effect on the algorithmic efficiency (thus, reducing the number of serial processed function evaluation) resembles a nearly linear behaviour, which appears to be particularly robust for parallelization factors below 30. This behaviour indicates that parallelization will further significantly improve performance even for much larger parallelization factors. In regard to the concrete set of 200 instances, it can be seen from Figure 3 and Table 2, that a parallelization factor of 300 makes the MIDACO algorithm *over 150 times* more effective as in serial mode. In other words: Using a parallelization factor of 300 reduces the number of (serial processed) function evaluation by a factor of 150.

3.1 Additional Numerical Results

In addition to the previously presented numerical results investigating the parallelization effect on MIDACO, a separate numerical test run investigating MIDACO's capability to locate global optimal solutions is shown here. In contrast to previous numerical runs, which considered 100 executions applying a maximal function evaluation budget of 10 millions to each problem, a single (unparalleled) run on the full library with a time limit of 3 hours (10080 seconds) for each problem instance is considered here. Purpose of this additional numerical run is to evaluate the fundamental potential of MIDACO to solve even the harder instances of the test bed. Again, the lower bounds of each problem instance were used as starting point and the original bounds were considered for each problem. All MIDACO parameter were set to default.

Table 3. MIDACO performance on 200 MINLP's

Number of problems in total:	200
Number of optimal solutions:	172
Number of feasible solutions:	194
Average evaluation:	7,548,745
Average CPU-time:	1,635.3 sec
Total CPU-time:	3.7 days

Table 3 lists a summary of the results obtained by MIDACO on the full library. Note that the execution of this test run took 3.7 days of CPU-time.

From Table 3 it can be seen that MIDACO is able to obtain in 194 out of 200 cases a feasible solution. Out of this 194 feasible solutions, 172 solutions were globally optimal. The average number of function evaluation took around 7.5 million which where processed in about half an hour (1635 sec) on average.

The Appendix lists the individual MIDACO results on each of the 200 MINLP instances. In regard to the number of variables, MIDACO is able to solve the largest instance in the set (see benchmark "PARALLEL" in Table 5) to global optimality in about 15 minutes. This "PARALLEL" instance considered 205 variables and 115 constraints, including 81 equality constraints. Other large instances that could be solved to global optimality include "M7" with 114 variables and 211 constraints or "MINLPHIX" with 84 variables and 92 constraints. Several instances with over a hundred varibales and/or constraints are solved to a feasible but not global optimal solution, see e.g. benchmark M6, ST_E31, RAVEM and EX1244. Note that the majority of problems containing only few variables and/or constraints are solved to global optimality *within less than 0.05 seconds* and most instances with ten's of variables and/or constraints are solved within few seconds or even below a second.

4 Conclusion

A numerical assessment of the performance scalability of an evolutionary optimization algorithm on a set of 200 mixed integer nonlinear programming instances was presented. The MIDACO optimization software was chosen to represent the evolutionary algorithm as it offers mixed-integer capability combined with a seamless parallelization feature (see Section 2.1). In Section 3 it was demonstrated, that the performance in obtaining global optimal solutions can be significantly improved by evaluating blocks of solution candidates in parallel. Performance was understood here in a reduction of serial processed blocks, while maintaining a similar high number of optimal solutions. In Section 3 (Table 2) it was shown that for a parallelization factor of **P**=300 the MIDACO performance could be improved *over 150 times* in comparison to its unparallelized behaviour. As many real-world applications are CPU-time intensive, the required number of serial processed blocks often marks the bottleneck in optimizing such applications. Hence the presented results are especially relevant to this kind of CPU-time intensive real-world applications.

Another interesting finding of this study concerns the scale-up behaviour observed. From Figure 3 in Section 3 it could be seen that the scale-up effect resembles a nearly linear behaviour, whereas especially for low parallelization factors (less than 30) the efficiency gain was close to its theoretical maximum. Such behaviour implies that the algorithm will further significantly benefit from even much larger parallelization factors. Given that parallelization is a growing trend in CPU-architecture, this observation is encouraging.

Table 4. Abbreviations used in Table 5

Abbreviation	Description
Name	Name of the benchmark instance
n	Number of variables (in total)
n_i	Number of integer variables
m	Number of constraints (in total)
m_e	Number of equality constraints
Time	Amount of CPU-time (in seconds)
Status:	Solution status obtained by MIDACO
$\surd$	Status = Global optimum reached
-	Status = Feasible local solution
x	Status = Infeasible solution

Appendix

This Appendix lists all 200 MINLP benchmark instances with their name and number and type of variables and constraints in Table 5. It furthermore reports the MIDACO (5.0 beta version) default performance with a maximal CPU-time budget of 3 hours (10080 seconds). For detailed information on the global optimality criteria see Section 3. Note that in two benchmark cases (ST_TEST1

and WU_4) the global optimal solution lies on the lower bounds and hence equals the starting point, which implies a reported single evaluation in Table 5. Note that in Schlueter and Munetomo [23] a enlarged version of this appendix can be found, additionally listing the number of function evaluation for each benchmark.

Table 5. MINLP benchmarks results

Benchmark Details					MIDACO Result	
Name	n	n_i	m	m_e	Time	Status
MITP1	5	3	1	0	0.0	√
MITP2	5	3	7	0	0.0	√
QIP1	4	4	4	0	0.0	√
ASAADI11	4	3	3	0	0.0	√
ASAADI12	4	4	3	0	0.0	√
ASAADI21	7	4	4	0	0.0	√
ASAADI22	7	7	4	0	0.0	√
ASAADI31	10	6	8	0	0.0	√
ASAADI32	10	10	8	0	0.0	√
DIRTY	25	13	10	0	0.0	√
BRAAK1	7	3	2	0	0.0	√
BRAAK2	7	3	4	0	0.0	√
BRAAK3	7	3	4	0	0.0	√
DEX2	2	2	2	0	0.0	√
FUEL	15	3	15	6	2.0	√
WP02	2	1	2	0	0.0	√
NVS01	3	2	3	1	0.1	√
NVS02	8	5	3	3	0.1	√
NVS03	2	2	2	0	0.0	√
NVS04	2	2	0	0	0.0	√
NVS05	8	2	9	4	10800.0	-
NVS06	2	2	0	0	0.0	√
NVS07	3	3	2	0	0.0	√
NVS08	3	2	3	0	0.0	√
NVS09	10	10	0	0	0.0	√
NVS10	2	2	2	0	0.0	√
NVS11	3	3	3	0	0.0	√
NVS12	4	4	4	0	0.0	√
NVS13	5	5	5	0	0.0	√
NVS14	8	5	3	3	0.0	√
NVS15	3	3	1	0	0.0	√
NVS16	2	2	0	0	0.0	√
NVS17	7	7	7	0	0.0	√
NVS18	6	6	6	0	0.0	√
NVS19	8	8	8	0	0.0	√

Table 6. MINLP benchmarks results (continued)

Benchmark Details					MIDACO Result	
Name	n	n_i	m	m_e	Time	Status
NVS20	16	5	8	0	0.1	√
NVS21	3	2	2	0	0.0	√
NVS22	8	4	9	4	1047.3	√
NVS23	9	9	9	0	0.0	√
NVS24	10	10	10	0	0.0	√
GEAR	4	4	0	0	0.0	√
GEAR2	28	24	4	4	0.1	√
GEAR2A	28	24	4	4	0.2	√
GEAR3	8	4	4	4	0.0	√
GEAR4	6	4	1	1	0.1	√
M3	26	6	43	0	5.1	√
M6	86	30	157	0	10800.0	-
M7	114	42	211	0	10514.2	√
FLOUDAS1	5	3	5	2	0.0	√
FLOUDAS2	3	1	3	0	0.0	√
FLOUDAS3	7	4	9	0	0.0	√
FLOUDAS4	11	8	7	3	3.2	√
FLOUDAS40	11	8	7	3	0.0	√
FLOUDAS5	2	2	4	0	0.0	√
FLOUDAS6	2	1	3	0	0.0	√
SPRING	17	12	8	5	0.0	√
DU_OPT5	20	13	9	0	0.1	√
DU_OPT	20	13	9	0	0.3	√
ST_E13	2	1	2	0	0.0	√
ST_E14	11	4	13	4	0.1	√
ST_E15	5	3	5	2	0.0	√
ST_E27	4	2	6	0	0.0	√
ST_E29	11	8	7	2	0.9	√
ST_E31	112	24	135	81	10800.0	-
ST_E32	35	19	18	17	1876.8	√
ST_E35	32	7	39	15	9.6	√
ST_E36	2	1	2	1	0.0	√
ST_E38	4	2	3	0	0.0	√
ST_E40	4	3	8	4	0.0	√
ST_MIQP1	5	5	1	0	0.0	√

Table 7. MINLP benchmarks results (continued)

Benchmark Details					MIDACO Result	
Name	n	n_i	m	m_e	Time	Status
ST_MIQP2	4	4	3	0	0.0	√
ST_MIQP3	2	2	1	0	0.0	√
ST_MIQP4	6	3	4	0	0.1	√
ST_MIQP5	7	2	13	0	0.1	√
ST_TEST1	5	5	1	0	0.0	√
ST_TEST2	6	6	2	0	0.0	√
ST_TEST3	13	13	10	0	0.0	√
ST_TEST4	6	6	5	0	0.0	√
ST_TEST5	10	10	11	0	0.0	√
ST_TEST6	10	10	5	0	0.0	√
ST_TEST8	24	24	20	0	0.7	√
TESTGR1	10	10	5	0	0.0	√
TESTGR3	20	20	20	0	0.0	√
TESTPH4	3	3	10	0	0.0	√
TLN2	8	8	12	0	0.0	√
TLN4	24	24	24	0	0.1	√
TLN5	35	35	30	0	0.6	√
TLN6	48	48	36	0	6.6	√
NEJI	3	1	6	0	0.0	√
TST_NAG	8	4	7	2	10800.0	x
TLOSS	48	48	53	0	10800.0	-
TLTR	48	48	54	0	0.2	√
MEANVARX	35	14	44	8	0.3	√
MINLPHIX	84	20	92	30	1722.2	√
MIP_EX	5	3	7	0	0.0	√
MGRID_C1	5	5	1	0	0.0	√
MGRID_C2	10	10	1	0	0.0	√
CROP5	5	5	3	0	0.0	√
CROP20	20	20	3	0	0.1	√
CROP50	50	50	3	0	0.1	√
CROP100	100	100	3	0	4.3	√
SPLITF1	12	9	9	3	0.0	√
SPLITF2	24	18	15	6	0.0	√
SPLITF3	24	18	15	6	0.3	√
SPLITF4	24	18	15	6	0.1	√
SPLITF5	24	18	15	6	0.4	√

Table 8. MINLP benchmarks results (continued)

Benchmark Details					MIDACO Result	
Name	n	n_i	m	m_e	Time	Status
SPLITF6	24	18	15	6	0.0	√
SPLITF7	36	27	21	9	140.8	√
SPLITF8	36	27	21	9	0.4	√
SPLITF9	36	27	21	9	1.0	√
ELF	54	24	38	6	9.2	√
SPECTRA2	69	30	72	9	10800.0	-
WINDFAC	14	3	13	13	56.4	√
CSCHED1	76	63	22	12	10800.0	-
ALAN	8	4	7	2	0.2	√
PUMP	24	9	34	13	10800.0	-
RAVEM	112	54	186	25	10800.0	-
ORTEZ	87	18	74	24	10800.0	-
EX1221	5	3	5	2	0.0	√
EX1222	3	1	3	0	0.0	√
EX1223	11	4	13	4	0.5	√
EX1223A	7	4	9	0	0.0	√
EX1223B	7	4	9	0	0.0	√
EX1224	11	8	7	2	0.8	√
EX1225	8	6	10	2	0.0	√
EX1226	5	3	5	1	0.0	√
EX1233	52	12	64	20	10800.0	-
EX1243	68	16	96	24	10800.0	-
EX1244	95	23	129	30	10800.0	-
EX1252	39	15	43	22	10800.0	-
EX1263	92	72	55	20	10800.0	-
EX1263A	24	24	35	0	0.7	√
EX1264	88	68	55	20	10800.0	-
EX1264A	24	24	35	0	3.0	√
EX1265	130	100	74	30	10800.0	-
EX1265A	35	35	44	0	2.0	√
DIOPHE	4	4	1	1	0.0	√
EX1266A	48	48	53	0	10800.0	-
GBD	4	3	4	0	0.0	√
EX3	32	8	31	17	41.0	√
EX4	36	25	30	0	0.5	√
FAC1	22	6	18	10	0.3	√

Table 9. MINLP benchmarks results (continued)

Benchmark Details					MIDACO Result	
Name	n	n_i	m	m_e	Time	Status
FAC2	66	12	33	21	9.2	√
FAC3	66	12	33	21	8.8	√
GKOCIS	11	3	8	5	0.0	√
KG	9	2	9	5	0.2	√
SYNTHES1	6	3	6	0	0.0	√
SYNTHES2	11	5	14	1	0.0	√
SYNTHES3	17	8	23	2	0.3	√
PARALLEL	205	25	115	81	1159.4	√
SYNHEAT	56	12	64	20	10800.0	-
SEP1	29	2	31	22	25.6	√
DAKOTA	4	2	2	0	0.0	√
BATCH	47	24	73	12	36.9	√
BATCHDES	19	9	19	6	0.0	√
ENIPLAC	141	24	189	87	10800.0	x
PROB02	6	6	8	0	0.0	√
PROB03	2	2	1	0	0.0	√
PROB10	2	1	2	0	0.0	√
NOUS1	50	2	43	41	10800.0	-
NOUS2	50	2	43	41	10800.0	-
TLS2	37	33	24	6	2.1	√
TLS4	105	89	64	20	10800.0	-
TLS5	161	136	90	30	10800.0	-
OAER	9	3	7	3	0.0	√
PROCSEL	10	3	7	4	0.5	√
LICHOU_1	2	1	2	1	0.0	√
LICHOU_2	4	2	4	0	0.0	√
LICHOU_3	3	3	4	0	0.0	√
WU_1	32	32	0	0	0.0	√
WU_2	32	32	0	0	0.0	√
WU_3	64	64	0	0	0.0	√
WU_4	64	64	0	0	0.0	√
OPTPRLOC	30	25	30	0	0.0	√
GASNET	90	10	69	48	10800.0	x

Table 10. MINLP benchmarks (continued)

Benchmark Details					MIDACO Result	
Name	n	n_i	m	m_e	Time	Status
TP83	5	4	6	0	0.0	√
TP84	5	2	6	0	0.0	√
TP85	5	3	38	0	0.0	√
TP87	6	2	4	4	0.0	√
TP93	6	1	2	0	0.1	√
FEEDTRAY	97	7	91	83	10800.0	x
FEEDTRAY2	87	36	283	6	10800.0	x
HILBERT20	20	20	20	20	2.0	√
HILBERT50	50	50	50	50	1084.9	√
HILBERT100	100	100	100	100	6872.4	√
SLOPPY	6	6	3	0	0.0	√
RASTRIGIN	2	1	0	0	0.0	√
EMSO	6	3	4	0	0.0	√
TP1	2	2	0	0	0.0	√
TP1A	2	2	0	0	0.0	√
TP1B	2	2	0	0	0.0	√
TP9	2	2	1	1	0.0	√
TP10	2	2	1	0	0.0	√
DEB10	182	22	129	65	10800.0	x
IRAP1	68	68	18	0	1.7	√
IRAP2	38	38	20	0	0.0	√
IRAP3	40	40	21	0	0.0	√
IRAP4	45	45	16	0	1.1	√
IRAP5	60	60	16	0	2.5	√
IRAP6	34	34	18	0	0.0	√

References

[1] Babu B., Angira A., A differential evolution approach for global optimisation of minlp problems, In: Proceedings of the Fourth Asia Pacific Conference on Simulated Evolution and Learning (SEAL 2002), Singapore, 2002, pp. 880–884.

[2] Cardoso M.F., Salcedo R.L., Azevedo S.F., Barbosa D., A simulated annealing approach to the solution of MINLP problems, Computers Chem. Engng. 12(21), 1997, pp. 1349–1364.

[3] Costa L., Oliveira P., Evolutionary algorithms approach to the solution of mixed integer nonlinear programming problems, Comput Chem Eng, 25(23), 2001, 257-266.

[4] Deep K., Krishna P.S., Kansal M.L., Mohan C., A real coded genetic algorithm for solving integer and mixed integer optimization problems. Appl. Math. Comput., 212(2), 2009, pp. 505–518.

[5] European Space Agency (ESA) and Advanced Concepts Team (ACT), Gtop database - global optimisation trajectory problems and solutions, Software available at http://www.esa.int/gsp/ACT/inf/op/globopt.htm, 2011.

[6] Glover F., Parametric tabu-search for mixed integer programs, Comput Oper Res 33(9), 2006, 24492494.

[7] Gupta S., Tan G., A scalable parallel implementation of evolutionary algorithms for multi-objective optimization on GPUs, Evolutionary Computation (CEC), IEEE Congress on, Sendai, 2015, pp. 1567–1574.

[8] Quinn J.M., Parallel Programming in C with MPI and OpenMP, McGraw-Hill, 2003.

[9] GAMS MINLPlib - A collection of Mixed Integer Nonlinear Programming models. Washington, DC, USA; software available at http://www.gamsworld.org/minlp/minlplib.htm, 2016.

[10] Laessig J., Sudholt D., General upper bounds on the runtime of parallel evolutionary algorithms, Evolutionary Computation, vol. 22, no. 3, 2014, pp. 405-437.

[11] Liang B., Wang J., Jiang Y., Huang D., Improved Hybrid Differential Evolution-Estimation of Distribution Algorithm with Feasibility Rules for NLP/MINLP, Engineering Optimization Problems, Chin. J. Chem. Eng. 20(6), 2012, pp. 1074–1080.

[12] Mohamed A.W., An efficient modified differential evolution algorithm for solving constrained nonlinear integer and mixed-integer global optimization problems. Int. J. Mach. Learn. & Cyber., 2015, pp. 1–19.

[13] Munawar A., Redesigning Evolutionary Algorithms for Many-Core Processors Ph.D. Thesis, Graduate School of Information Science and Technology, Hokkaido University, Japan, 2012.

[14] Du X., Ni Y., Yao Z., Xiao R., High performance parallel evolutionary algorithm model based on MapReduce framework, Int. J. Computer Applications in Technology, Vol. 46, No. 3, 2013, pp. 290–296.

[15] Powell D., Hollingsworth J., A NSGA-II, web-enabled, parallel optimization framework for NLP and MINLP, Proceedings of the 9th annual conference on Genetic and evolutionary computation, 2007, pp. 2145–2150.

[16] Sakuray Pais M., Yamanaka K., Rodrigues Pinto E., Rigorous Experimental Performance Analysis of Parallel Evolutionary Algorithms on Multicore Platforms, In IEEE Latin America Transactions, vol. 12, no. 4, 2014, pp. 805–811.

[17] Schlueter M., Egea J.A., Banga J.R., Extended ant-colony optimization for non-convex mixed integer nonlinear programming, Comput. Oper. Res. 36(7), 2009, 2217–2229.

[18] Schlueter M., Gerdts M., The Oracle Penalty Method. J. Global Optim. 47(2), 2010, 293–325.

[19] Schlueter, M., Gerdts, M., Rueckmann J.J., A Numerical Study of MIDACO on 100 MINLP Benchmarks, Optimization 7(61), 2012, pp. 873–900.

[20] Schlueter M., Erb S., Gerdts M., Kemble S., Rueckmann J.J., MIDACO on MINLP Space Applications, Advances in Space Research, 51(7), 2013, 1116–1131.

[21] Schlueter M., MIDACO Software Performance on Interplanetary Trajectory Benchmarks, Advances in Space Research, 54(4), 2014, 744–754.

[22] Schlueter M., MIDACO Solver - Global Optimization Software for Mixed Integer Nonlinear Programming, Software available at http://www.midaco-solver.com, 2016.

[23] Schlueter M., Munetomo M., Numerical Assessment of the Parallelization Scalability on 200 MINLP Benchmarks, Proc. of the IEEE-CEC2016 Conference, Vancouver, Canada, 2016.

[24] K. Schittkowski, A Collection of 200 Test Problems for Nonlinear Mixed-Integer Programming in Fortran (User Guide), Report, Department of Computer Science, University of Bayreuth, Bayreuth, 2012.

[25] K. Schittkowski, NLPQLP - A Fortran implementation of a sequential quadratic programming algorithm with distributed and non-monotone line search (User Guide), Report, Department of Computer Science, University of Bayreuth, Bayreuth, 2009.

[26] K. Socha and M. Dorigo, Ant colony optimization for continuous domains, Eur. J. Oper. Res. 85, 2008, pp. 1155–1173.

[27] Sudholt D., Parallel Evolutionary Algorithms, In Janusz Kacprzyk and Witold Pedrycz (Eds.): Handbook of Computational Intelligence, Springer, 2015.

[28] Wasanapradit T., Mukdasanit N., Chaiyaratana N., Srinophakun T., Solving mixed-integer nonlinear programming problems using improved genetic algorithms, Korean J. Chem. Eng. 28(1), 2011, 32–40.

[29] Yiqing L., Xigang Y., Yongjian L., An improved PSO algorithm for solving non-convex NLP/MINLP problems with equality constraints, Comp. Chem. Eng. 3(31), 2007, 153–162.

[30] Young C.T., Zheng Y., Yeh C.W., Jang S.S., Information-guided genetic algorithm approach to the solution of MINLP problems, Ind. Eng. Chem. Res. 46, 2007, pp. 1527–1537.

[31] Yingyong Z., Yongde Z., Qinghua L., Jingang J., Guangbin Y., Improved Multi-objective Genetic Algorithm Based on Parallel Hybrid Evolutionary Theory, International Journal of Hybrid Information Technology Vol.8, No.1, 2015, pp. 133–140.

[32] Yue T., Guan-Zheng T., Shu-Guang D., Hybrid particle swarm optimization with chaotic search for solving integer and mixed integer programming problems. J. Central South Univ., 2014, 21:2731–2742.

Martin Schlueter is a post-doctoral researcher at the Information Initiative Center, Hokkaido University, Japan. He obtained his Ph.D. from the School of Mathematics at the University of Birmingham (UK) in 2012. In collaboration with the European Space Agency (ESA) and EADS-Astrium he developed the MIDACO optimization algorithm, which holds several best-known record solutions to challenging interplanetary space trajectory benchmarks. From 2014 to 2016 he was appointed as research scientist at the Institute of Space and Astronautical Science (ISAS) of the Japan Aerospace Exploration Agency (JAXA) to investigate multi-objective optimization via massively parallelized evolutionary algorithms.

Masaharu Munetomo is professor and vice director of Information Initiative Center, Hokkaido University, which is one of the Japanese national supercomputing center also providing cloud computing services to researchers. He recieved Ph.D. in information engineering on 1996. He joined IlliGAL (Illinois Genetic Algorithms Laboratory), University of Illinois at Urbana-Champaing as a visiting scholar from 1998 to 1999. He is the chief architect of "Hokkaido University Academic Cloud" which started services in 2011 as the largest academic cloud system in Japan. He is the chief examinar of cloud computing research group of Japanese national supercomputing centers. Currently, he conduct research projects related to evolutionary computation, metaheuristics, machine learning, and cloud computing.