# A Monte Carlo Iterative Optimization Algorithm for Integer Linear Programming Problems

**Takeshi Tengan[†], Takeo Yoshida[††], and Morikazu Nakamura[††]**

[†]Faculty of International Studies, Meio University, Nago, Okinawa, 905-8585 Japan
[††]Faculty of Engineering, University of the Ryukyus, Nishihara, Okinawa, 903-0213 Japan

**Summary**
In this paper, we present a new optimization technique for integer linear programming problems. The proposed method is a metaheuristic algorithm and improves solutions by iterating the problem reduction and solving the reduced problem. The algorithm is a hybrid approach in which we use a Metropolis-Hastings algorithm and an exact solver and has a good characteristic such that the reduced problem at each iteration has better or equal quality feasible solutions. The experimental evaluation shows that our method can obtain a good quality of solutions within reasonable execution time for hard problems specified in MIPLIB2010.

***Key words:***
*Integer Programming, Iterative Optimization, Problem Reduction, Monte Carlo Method, Metaheuristics*

## 1. Introduction

Combinatorial optimization problems have been widely studied in computer sciences and operations researches for theoretical and practical interests. Their importance becomes larger and larger since our big data society encourages us to use data generated in and collected from various fields and combinatorial optimization is an essential technique to extract valuable information from received data [1]. For example, we can formulate many problems in cyber-physical systems where we try to obtain useful information from sensing data collected in physical systems. Bioinformatics is also well-known application areas where many problems correspond to combinatorial optimization problems such as genome assembling, multiple alignments, and motif identification in as optimization problems [2].

For NP-hard combinatorial optimization problems, there exist two main approaches, the exact algorithm and the approximation one. The exact algorithm, the branch and bound algorithm, the branch and cut, find a global optimum solution for a given instance [3]. Thanks to the drastic improvement of searching algorithms for the integer linear programming and computing platforms, we can solve precisely many practical problems by using solvers such as Gurobi optimizer [4] and CPLEX [5]. Therefore, the mathematical programming approach has been practicable for many application areas not only in engineering [6], [7], [8], [9] but also in agriculture [10], [11], biology, even sociology.

However, there are two major obstacles for many users to receive benefits from the mathematical programming. One is the difficulty of mathematical programming formulation; users need to formulate problems as integer programming problems, which requires in-depth knowledge on the related mathematics and empirical skills to represent problems as a class of the integer programming, hopefully, integer linear programming. The other obstacle is high possibility of unreasonable computation time, that is, its NP-hardness should require long computation time beyond extraordinarily our expectation.

For the former, some ideas to reduce the difficulty of the mathematical formulation are proposed in [3]. We also introduced a Petri net approach to generate mixed integer linear programming problems [12]. For the latter, we need to change from the exact solving to the approximation. There are some approaches; the approximation algorithms with provable solution quality for specific problems, heuristic, and metaheuristic approaches are without guaranteeing solution quality.

Exact algorithms and approximation algorithms have been independently studied for a long time, however, recently a combination of both approaches is studied since they can complement each other. For example, to calculate good incumbent solutions or efficient bounds [13]. In other studies, exact algorithms are utilized to calculate the best solution in large neighborhood structure [14] and to enhance the decoding part, the branch & bound algorithm is used in GAs [15], [16].

In our previous work [17], we proposed hybrid algorithms that combine an exact algorithm and metaheuristic where the exact algorithm is used for the solution evaluation in the metaheuristics to realize robustness for the large-scale combinatorial optimization. That is, our method can obtain a good quality of solutions within a reasonable time for almost all instances.

In this paper, we formulate our algorithm as an iterative optimization method and perform an experimental evaluation of our approach because we had shown just first data of preliminary evaluation in [17] without solving hard problems.

Manuscript received November 5, 2018
Manuscript revised November 20, 2018

Table 1: Constraint Types

| | Type | Mathematical Form |
|---|---|---|
| **AGG** | Aggregation | $a_i x_i + a_k x_k = b, x_i \in \mathbb{N}$ or $\mathbb{R}, a_i, a_k, b \in \mathbb{R}$ |
| **VBD** | Variable Bound | $x_i \leq a_k x_k + b$ or $x_i \geq a_k x_k + b, x_i, x_k$ int. or cont., $a_k, b \in \mathbb{R}$ |
| **PAR** | Set Partition | $\sum x_i$=1, $x_i \in \{0,1\}$ |
| **PAC** | Set Packing | $\sum x_i \leq$1, $x_i \in \{0,1\}$ |
| **COV** | Set Cover | $\sum x_i \geq$1, $x_i \in \{0,1\}$ |
| **CAR** | Cardinality | $\sum x_i$=$b$, $x_i \in \{0,1\}, b \in \mathbb{Z}$ |
| **EQK** | Equality Knapsack | $\sum a_i x_i$=$b$, $x_i \in \{0,1\}, a, b \in \mathbb{N}$ |
| **BIN** | Bin Packing | $\sum a_i x_i + a_k x_k$=$a_k$, $x_i \in \{0,1\}, a_i, a_k \in \mathbb{N}$ |
| **IVK** | Invariant Knapsack | $\sum x_i \leq b$, $x_i \in \{0,1\}, b \in \mathbb{N}$ |
| **KNA** | Knapsack | $\sum a_i x_i \leq b$, $x_i \in \{0,1\}, a_i, b \in \mathbb{N}$ |
| **IKN** | Integer Knapsack | $\sum a_i x_i \leq b$, $x_i \in \mathbb{Z}, a_i, b \in \mathbb{N}$ |
| **M01** | Mixed Binary | $\sum a_i x_i + \sum p_j s_j \leq (\text{or} =)\ b$, $x_i \in \{0,1\}, s_j$ cont., $a_i, p_j, b \in \mathbb{R}$ |

## 2. Integer Linear Programming Problem

Integer linear programming problems are a class of mathematical programming problems in which some of all the variables restricted to be integer and all the constraints and the objective function should be denoted as linear equations or inequations. An integer linear programming problem in the canonical form is expressed as follows:

Objective:

$$\max \sum_{i=1}^{N} c_i x_i \quad (1)$$

s.t.

$$\sum_{i=1}^{N} a_{j,i} x_i \leq b_i, \forall j \in \{1, \ldots, M\} \quad (2)$$

$$x_i \geq 0, \forall i \in \{1, \ldots, N\} \quad (3)$$

$$x_i \in \mathbb{N} \quad (4)$$

The formulation includes two types of constraints; Constraints (1) and (2) are called Integer Knapsack (IKN) and Variable Bound (VBD), respectively.

Table 1 shows well-known constraint types [18]. We cannot say easily which types of constraints make problem instances difficult but they give us beneficial information.

## 3. Monte Carlo Iterative Optimization

This section proposes an iterative optimization algorithm which enables us to obtain good heuristic solutions for integer linear programming problems. Recent drastic advancement of the mathematical programming research partially overcomes the intractableness of NP-hard problems, that is, current powerful solvers can solve integer linear programming problems of practical sizes in reasonable time. However, the exact algorithms cannot be a universal approach which can treat any types of integer linear programming problems. For problem instances with a massive number of variables and/or complex solution spaces, we cannot solve NP-hard problems in reasonable time.

Our algorithm is inspired from the EM algorithm [19], the Expectation-Maximum algorithm in statistics, in a sense that the EM algorithm iteratively estimates parameters in statistical models and maximizes the expected likelihood in alternately performed E(Expectation)-step and M(Maximization) step. Our algorithm is also an iterative optimization in which it finds a set of variables to be removed practically from the original problem, R(Reduction)-step), and solves the reduced problem by removing the set of variables at each iteration, O(Optimization)-step.

### 3.1 Problem Reduction

We consider here the integer linear programming problem shown in the previous section, where the variables are an integer, and the types of constraints are only IKN and VBD in Table 1. We can easily apply the discussion here to other cases, binary variables, and different kinds of constraints.

The solution space of mathematical programming problems widens with the number of variables. The computation time for the optimization depends not only on the solution space size but also a combination of constraints since the combination of constraints determines the complexity of the solution landscape. The computation time can be extremely different when constraints are different even though the number of variables is the same between two problem instances. However, the number of variables is still an essential factor to represent the hardness for exact solving. Therefore, we use the number of variables to show the ratio of the problem reduction.

In our algorithm, the problem reduction is performed by fixing variables with values as new constraints. Let us consider the following integer linear programming problem,

where we just show the problem explained in Section 2 in a matrix form.

$f(\boldsymbol{X})$:
$\max \boldsymbol{CX}$ (5)
$\boldsymbol{AX} \leq \boldsymbol{B}$ (6)
$X_i \geq 0, \forall i \in \{1, \dots, N\}$ (7)
$X_i \in \mathbb{N}, \forall i \in \{1, \dots, N\}$ (8)

By introducing the constraints specified in $\theta_X$, we obtain a reduced problem for $f(\boldsymbol{X})$. Let us denote the reduced problem by $f(X|\theta_X)$.

$f(\boldsymbol{X}|\theta_X)$:
$\max \boldsymbol{CX}$ (9)
$\boldsymbol{AX} \leq \boldsymbol{B}$ (10)
$X_i = x_i, \forall (X_i = x_i) \in \theta_X$ (11)
$X_i \geq 0, \forall i \in \{1, \dots, N\}$ (12)
$X_i \in \mathbb{N}, \forall i \in \{1, \dots, N\}$ (13)

where $x_i$ is a feasible value for variable $X_i$. Note that the number of variables in the reduced problem is, in practice, $|\boldsymbol{X}| - |\theta_X|$.

## 3.2 Reduction and Optimization Steps

We present a new iterative algorithm to solve heuristically integer linear programming problems. As we explained above, the algorithm is based on the EM algorithm, well-known for maximum likelihood methods.
Our algorithm iteratively generate fixed variable sets, $\theta_X^{(0)}, \theta_X^{(1)}, \dots, \theta_X^{(k)}$, and at each iteration solve the reduced problem, $f(\boldsymbol{X}|\theta_X^{(i)})$, $i = 0, 1, \dots, k$, where $\theta_X^{(i)}$ is determined from a solution of problem $f(\boldsymbol{X}|\theta_X^{(i-1)})$, and $|\theta_X^{(i)}| < |\boldsymbol{X}|$. The proposed algorithm is composed of two steps:

**R-Step:** Reduce the size of the integer linear optimization problem $f(\boldsymbol{X})$ by adding a constraint set $\theta_X^{(i)}$ and generate the reduced problem $f(\boldsymbol{X}|\theta_X^{(i)})$.
**O-Step:** Solve the reduced problem $f(\boldsymbol{X}|\theta_X^{(i)})$ by some optimization algorithm.

For O-Step, we can utilize any solver. In our implementation, we used Gurobi Optimizer, one of the well-known powerful commercial solvers. For the reduced problem, we can obtain exact solutions.
In R-Step, we need to generate a reduced problem instance by introducing a set of constraints. To ensure that the reduced problem should have feasible solutions, we select a set of constraints for the problem reduction from a feasible solution.

Let $\boldsymbol{x}_o = (X_1 = x_1, X_2 = x_2, \dots, X_n = x_n)$ be a feasible solution of $f(\boldsymbol{X})$. We select some fixed number of variables and their values from $\boldsymbol{x}_o$ as a constraint set, for example, $\theta_X^{(0)} = \{X_3 = x_3, X_{10} = x_{10}, \dots\}$. Obviously, $f(\boldsymbol{X}|\theta_X^{(0)})$ has a feasible solution. And then, we obtain solution $\boldsymbol{x}^{(0)}$ by solving $f(\boldsymbol{X}|\theta_X^{(0)})$ and construct $\theta_X^{(1)}$ from $\boldsymbol{x}^{(0)}$ by selecting some variables and their values. This discussion can be extended to $\theta_X^{(i)}, i = 2, \dots, k$.

***Lemma 1:*** $f(\boldsymbol{X}|\theta_X^{(i)})$, i=0, 1, 2, …, *k*, has a feasible solution if $\boldsymbol{x}_o$ is feasible.
***Proof:*** We prove the lemma by induction. It is obvious that $f(\boldsymbol{X}|\theta_X^{(0)})$ has a feasible solution when $\boldsymbol{x}_o$ is feasible. We suppose that $f(\boldsymbol{X}|\theta_X^{(h)})$ has a feasible solution when $\boldsymbol{x}_o$ is feasible. Let us denote a solution of $f(\boldsymbol{X}|\theta_X^{(h)})$ by $\boldsymbol{x}^{(h)}$ and select a set of variables and their values for the next constraint set,
$\theta_X^{(h+1)} = \{X_i = x_i^{(h)} | \exists i \in \{1, \dots, N\},\ |\theta_X^{(h+1)}| < |\boldsymbol{X}|\}$. (14)
It is also obvious that $f(\boldsymbol{X}|\theta_X^{(h+1)})$ has a feasible solution. Q.E.D

The lemma gives us good property for our iterative algorithm.

***Theorem 1***: The optimum value of $f(\boldsymbol{X}|\theta_X^{(i)})$ is a lower bound for $f(\boldsymbol{X}|\theta_X^{(i+1)})$.
***Proof***: Let $\boldsymbol{x}^{(i)}$ be an optimal solution of $f(\boldsymbol{X}|\theta_X^{(i)})$. $\theta_X^{(i+1)}$ is constructed by selecting some elements in $\boldsymbol{x}^{(i)}$. From *Lemma 1*, $f(\boldsymbol{X}|\theta_X^{(i+1)})$ has a feasible solution and $\boldsymbol{x}^{(i)}$ is a known feasible solution. Therefore, the optimal solution of $f(\boldsymbol{X}|\theta_X^{(i+1)})$ cannot be worse than $\boldsymbol{x}^{(i)}$. Q.E.D

By Theorem 1, we confirm that the iteration of R-Step and O-Step cannot make the solution quality worse. However, from the practical point of view, the solution obtained by solving $f(\boldsymbol{X}|\theta_X^{(i)})$ can be worse. The reason for the contradiction is that the O-step cannot always obtain the exact solution of the reduced problem $f(\boldsymbol{X}|\theta_X^{(i)})$ in a reasonable time. This situation can be taken place in practice, that is, the solver takes a longer time than a prefixed TimeLimit. That is, the solution quality can be worse when the solver finds just a local optimum. Our approach can accept such worse solutions for making diversity in the searching process. This idea is the essential point in the Metropolis-Hastings algorithm or Simulated Annealing.

### 3.3 Monte Carlo Reduction & Optimization

Our iterative approach can improve the solution quality since *Theorem 1* ensures the monotonicity of solution improvement theoretically, even though we sometimes find worse solutions. We utilize the simplest and efficient way to generate a next fixed variable set, say *Monte Carlo* method. That is, we select randomly the next fixed variable set only based on the current situation, where we control the number of fixed variables.

We introduce a parameter Ratio to determine how many variables are reduced in the R-step.

$$Ratio = \frac{|\theta_X|}{|X|} \ (< 1.0) \tag{15}$$

Therefore, we set a large value for Ratio when we want to reduce the computation time of the exact algorithm since a larger value of Ratio leads to smaller size of the solution space. Note that the number of variables exponentially widens the solution space. Another parameter SC is to specify how many constraints are changed to select the next fixed variables from the current one at each iteration. Larger SC should be a long jump from the current solution, leading to diversification for searching. On the other hand, smaller SC can search more intensively high potential space.

Figure 1 shows a pseudo code of our algorithm, MonteCarloR&O. Function InitialFeasibleSolution() finds a feasible solution by using some algorithm. Note that the feasible solution may be far from the optimal solution. Function GenerateConstraintsFrom() determines the next fixed variable set from the previous solution and parameters Ratio and SC. Function Solve($f(\boldsymbol{X})$, TimeLimit) should return an optimal solution for $f(\boldsymbol{X})$. Note that the exact optimizer may require an extremely long time to get the exact solution even for reduced problems. To avoid such a situation, we set TimeLimit for function Solve(). Therefore, the function may return worse solutions than the previous solution. In this case, we utilize the Metropolis-Hastings algorithm to determine the acceptance of worse solutions at Lines 8 to 18.

```
 1: x_o ← InitialFeasibleSolution()
 2: θ_X^(0) ←GenerateConstraintsFrom(x_o, Ratio)
 3: CurSolution, x^(0) ← Solve(f(X|θ_X^(0)), TimeLimit)
 4: i ← 1
 5: repeat
 6:   θ_X^(i) ←GenerateConstraintsFrom(x^(i−1), Ratio, SC)
 7:   NewSolution, x^(i) ← Solve(f(X|θ_X^(i)), TimeLimit)
 8:   ΔSolution ← (CurSolution – NewSolution)
 9:   if ΔSolution < 0 then
10:     CurSolution ← NewSolution
11:     if BestSolution < NewSolution then
12:       x_best ← x^(i)
13:     endif
14:     i ← i+1
15:   else if RANDOM() < e^(−Δsolution/T) then
16:     CurSolution ← NewSolution
17:     i ← i+1
18:   endif
19: until Termination Condition holds
```

Fig. 1 Pseudo Code for MonteCarloR&O

## 4. Experimental Evaluation

We performed the experimental evaluation to show the effectiveness of our algorithm. We selected integer linear programming problems, characterized as Hard to solve, provided in MIPLIB2010 [1]. The selected problems from the library are shown in the first three rows in Table 2. The last one, the multiple knapsack problem, was used in our experiment to compare with traditional metaheuristics, SA since we can easily apply SA to the problem because of its simple constraints, but not other problems. In the experiment, we generate instances of the multiple knapsack problem randomly with varying problem size.

"Rows" and "Cols" denote the number of constraints and variables, respectively. "Constraint Types" corresponds to "Type" in Table 1. "Status" represents the difficulty of the problem. "Hard" means that the problem is quite difficult to solve.

Table 2: Test Problems in Experiments

| Name | Rows | Cols | Constraint Types | Status in MIPLIB2010 |
|---|---|---|---|---|
| **sct32** | 5,440 | 9,767 | *VBD, IVK, KNA, M01* | Hard |
| **mkc** | 3,411 | 5,325 | *VBD, PAC, IVK, M01* | Hard |
| **d10200** | 947 | 2,000 | *PAR, IVK, KNA* | Hard |
| **mkp** | Varied | Varied | *PAC, KNA* | N/A |

### 4.1 Effects of Problem Reduction

To confirm the effect of the problem reduction shown in Section 3.1, we investigated how the problem reduction effects on the computation time and the solution quality for three problem instances, sct32, mkc, and d10200. We varied Ratio from 0.0 to 0.4 in the experiment.
Table 3 and 4 show effects of the problem reduction on computation time and solution quality, respectively. In Table 4, the solution quality SQ was calculated by the following equation:

$$SQ = \frac{|(Exact\ \ Solution) - (Obtained\ \ Solution)|}{|(Exact\ \ Solution) - (Initial\ \ Feasible\ \ Solution)|}$$

where (Initial Feasible Solution) means the solution obtained firstly by Gurobi.
From the tables, we confirm that the execution time depends on strongly the reduction ratio and the solution quality is inversely proportional to the reduction ratio.

Table 3: Effects on Execution Time [s]

| Ratio [%] | sct32 | mkc | d10200 |
|---|---|---|---|
| **0** | 1,428,720 | 217,445 | N/A |
| **10** | 58,678 | 3,950 | 311,940 |
| **20** | 6,290 | 10 | 227,215 |
| **30** | 13 | 0 | 13,915 |
| **40** | 3 | 0 | 130 |

Table 4: Effects on Solution Quality SQ [%]

| Ratio [%] | sct32 | mkc | d10200 |
|---|---|---|---|
| **0** | 0 | 0 | 0 |
| **10** | 1.25 | 21.04 | 4.4 |
| **20** | 7.68 | 14.08 | 4.4 |
| **30** | 16.02 | 29.32 | 12.09 |
| **40** | 38.56 | 31.98 | 24.18 |

### 4.2 Performance Evaluation of MonteCalroR&O

To show the effectiveness of our algorithm, we evaluated by solving hard problems provided in MIPLIB2010, sct32, mkc, and d10200. In this experiment, we used parameter Ratio = 0.2, 0.1, 0.05 for sct32, mkc, d10200, respectively. These values are determined by considering the execution time should be less than two hours.
Table 5 shows the results. The execution time of Gurobi for mkc and d10200 was beyond 217,445 and 13,785, respectively. For our MonteCalroR&O, we varied SC, the number of swapped constraints, at each iteration from 1 to 30. SQ was calculated by the definition in Section 4.1.
From the results, our method can obtain good quality of solutions within reasonable computation time for the hard problems.

Table 5: Execution Time and Solution Quality for Hard Problems

| Name | *Gurobi* | MonteCalroR&O | | | |
|---|---|---|---|---|---|
| | Execution Time [s] | *Ratio* [%] | *SC* | Time [s] | *SQ* [%] |
| **sct32** | 1,428,720 | 20 | 1 | 5,509 | 2.57 |
| | | | 5 | 5,514 | 5.58 |
| | | | 10 | 2,046 | 17.78 |
| | | | 15 | 5,523 | 10.46 |
| | | | 20 | 5,526 | 8.86 |
| | | | 25 | 5,529 | 5.08 |
| | | | 30 | 5,537 | 11.55 |
| **mkc** | >217,445 | 10 | 1 | 5,520 | 19.53 |
| | | | 5 | 1,180 | 23.01 |
| | | | 10 | 322 | 16.92 |
| | | | 15 | 6,577 | 15.68 |
| | | | 20 | 1,467 | 24.38 |
| | | | 25 | 8,609 | 7.02 |
| | | | 30 | 6,203 | 15.40 |
| **d10200** | >13,785 | 5 | 1 | 977 | 7.69 |
| | | | 5 | 983 | 4.40 |
| | | | 10 | 990 | 5.49 |
| | | | 15 | 996 | 6.59 |
| | | | 20 | 1,004 | 8.79 |
| | | | 25 | 1,010 | 7.69 |
| | | | 30 | 1,015 | 2.20 |

We compared the solution quality with Simulated Annealing (SA). Note that it is difficult for other metaheuristics to search feasible solution spaces efficiently for arbitrarily given integer programming problems with many numbers and different kinds of constraints. Therefore, we utilized here the multiple knapsack problem (mkp in Table 2), a well-known and frequently-used benchmark problem instance, since the problem instances include only PAC and KNA constraints and they can be simply treated in metaheuristics.
Problem instances for the multiple knapsack problem are generated randomly with five knapsacks of different capacities varying the number of items from 100 to 2,000. For each instance, we solved once by Gurobi optimizer and twenty times by SA and our proposed method.
Figure 2 shows the comparison results. The horizontal axis represents the size of the multiple knapsack problem and the vertical one the ratio between the obtained solutions (each is the average of 20 runs) and the exact solutions. Therefore, closer values to 1 correspond to better quality solutions. We observed from the results that our proposed method was able to very good solutions for enough large problem instances, even though it is not better for small instances.

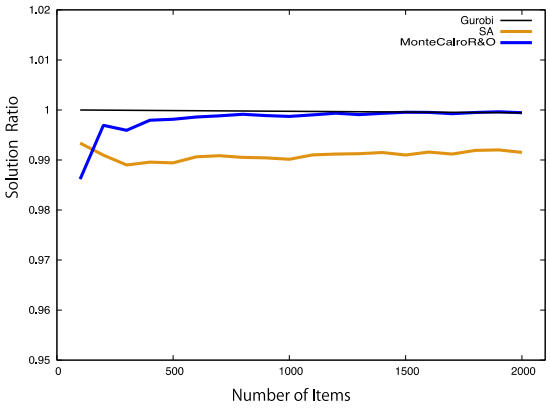


Fig. 2: Solution Quality Comparison between Gurobi, SA, and MonteCarloR&O

## 4.3 Parameters' Effects

As shown in Theorem 1, our method can improve solutions iteratively, but it should depend on parameters Ratio and SC. We investigated the effects of them on the improvement processes of our method.

Figures 3-6 show the improvement curves of the solutions with varying SC from 1 to 30 for Ratio 0.05, 0.1, 0.2, 0.3, respectively. From the figures, we observe that the solution curves are decreasing monotonously as Theorem 1 explains. Smaller Ratio leads to higher quality of solutions. However, we cannot always choose small Ratio because it requires much more expensive computational costs. Therefore, for a given Ratio, we need to choose carefully good SC, but it is not so easy since it depends on problem instance.

We consider a strategy so that we use first relatively large SC and gradually decreases it. Figure 7 depicts the improvement curves when we change SC at each iteration. Figure 7 shows also the SC pattern we used in the experiment. The SC pattern "30, 5, 5, 1, 1, 1" leads to better solution, compared to the other patterns and the static value cases. It means that our iterative approach has a potential to improve the performance by parameter control strategies.

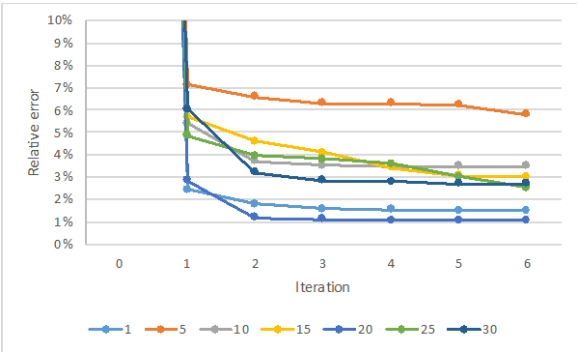


Fig. 3 Solution Improvements for Ratio 0.05

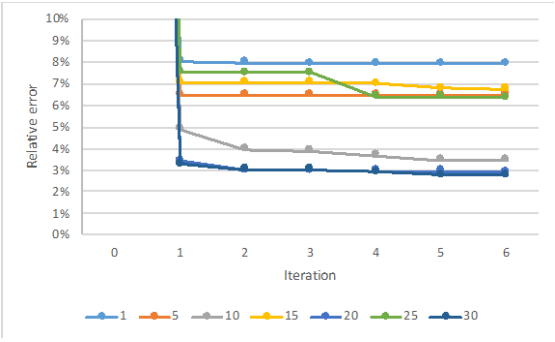


Fig. 4 Solution Improvements for Ratio 0.1

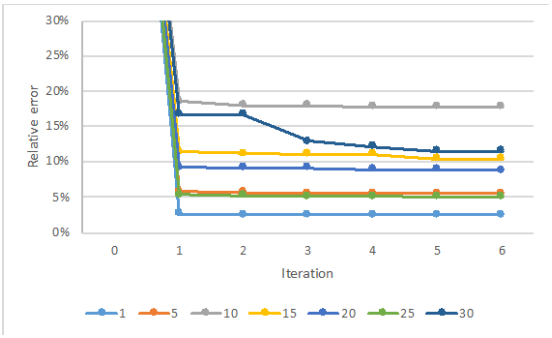


Fig. 5 Solution Improvements for Ratio 0.2

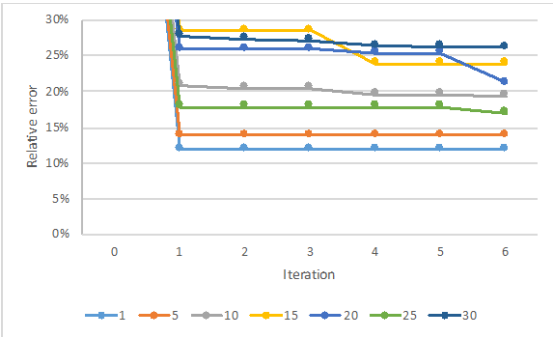


Fig. 6 Solution Improvements for Ratio 0.3

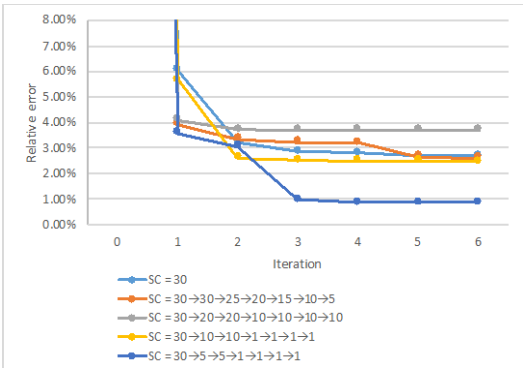


Fig. 7 Solution Improvements by Dynamic SC for Ratio 0.05

## 5. Concluding Remarks

In this paper, we presented a new optimization technique for integer linear programming problems. The proposed method, a metaheuristic algorithm, improves solutions by iterating the problem reduction and the optimization on the reduced problem. Our method has only two parameters, Ratio to balance between solution quality and computation time and SC between diversity and convergence.
The experimental evaluation showed that our method can obtain good quality of solutions within reasonable execution time for hard problems specified in MIPLIB2010. As future works, we need to develop strategies for dynamically changing parameters for the reduction size Ratio and the length of move SC according to the problem instance and a given upper limit of the computation time. Another work is to parallelize the method for speedup.

## References

[1] C. Dhaenens, L. Jourdan, Metaheuristics for Big Data, Willey, 2016.
[2] P. A. Pevzner, Computational Molecular Biology, An Algorithmic Approach, The MIT Press, Cambridge, 2000.
[3] H. Raul Williams, Model, Building in Mathematical Programming, 5th Edition, Wiley, 2013.
[4] GUROBI Optimizer, http://www.gurobi.com/
[5] CPLEX Optimizer, http://www-01.ibm.com/software/commerce/optimization/cplex-optimizer/
[6] Ahmet B. Keha, Ketan Khowala, John W. Fowler, "Mixed integer programming formulations for single machine scheduling problems," Computers & Industrial Engineering, Vol. 56, pp. 357-367, 2009.
[7] Jason Chao-Hsien Pana, Jen-Shiang Chenb, "Mixed binary integer programming formulations for the reentrant job shop scheduling problem," Vol. 32, Issue 5, pp.1197-1212, 2005.
[8] Debora P. Ronconi, Ernesto G. Birgin, "Mixed-integer programming models for flowshop scheduling problems minimizing the total earliness and tardiness," Just-in-Time Systems, Springer Optimization and Its Applications, pp. 91-105, 2012.
[9] Wen-Yang Ku, J. Christopher Beck, "Mixed integer programming models for job shop scheduling: A computational analysis," Computers & Operations Research, Vol. 73, pp.165-173, 2016.
[10] Senlin Guan, Morikazu Nakamura, Takeshi Shikanai, Takeo Okazaki, "Hybrid Petri nets modeling for farm work flow," Computers and Electronics in Agriculture, Vol. 62, No. 2, pp. 149-158, 2008.
[11] Senlin Guan, Morikazu Nakamura, Takeshi Shikanai, Takeo Okazaki, "Resource assignment and scheduling based on a two-phase metaheuristic for cropping system," Computers and Electronics in Agriculture, Vol. 62, No. 2, pp. 181-190, 2009.
[12] Andrea Veronica Porco, Ryosuke Ushijima, Morikazu Nakamura, "Automatic generation of mixed integer programming for scheduling problems based on colored timed Petri nets," vol. E101.A, no. 2, pp.367-372, 2018.
[13] D. L. Woodruff, "A chunking based selection strategy for integrating meta-heuristics with branch and bound," Metaheuristics: Advances and Trends in Local Search Paradigms for Optimization, pp. 499-511. Kluwer Academic Publishers, 1999.
[14] E. K. Burke, P. I. Cowling, and R. Keuthen, "Effective local and guided variable neighborhood search methods for the asymmetric travelling salesman problem," Applications of Evolutionary Computing: EvoWorkshops 2001, LNCS, vol. 2037, pp. 203-212. Springer, 2001.
[15] J. Puchinger, G. R. Raidl, and G. Koller, "Solving a real-world glass cutting problem," Evolutionary Computation in Combinatorial Optimization, EvoCOP 2004, LNCS, vol. 3004, pp. 162-173. Springer, 2004.
[16] A. T. Staggemeier, A. R. Clark, U. Aickelin, and J. Smith, "A hybrid genetic algorithm to solve a lot-sizing and scheduling problem," Proceedings of the 16th Triannual Conference of the International Federation of Operational Research Societies, Edinburgh, U.K., 2002
[17] K. Tamaki, T. Tengan, M. Nakamura, "Hybrid approaches based on simulated annealing and an exact algorithm for mixed integer programming problems," Proceedings of The Third International Conference on Networking and Computing, IEEE Press, 2012.
[18] Thorsten Koch, Tobias Achterberg, at el, "MIPLIB 2010 Mixed Integer Programming Library version 5," Mathematical Programming Computation, vol. 3, no. 2, pp.103-163, 2011.
[19] Dempster, A. P, "Maximum likelihood from incomplete data via the EM algorithm," Journal of the Royal Statistical Society Series B Methodological, Vol. 39, No.1, pp.1-38, 1977.

**Takeshi Tengan** received the B.E. and M.E degrees in Electronics and Information Engineering from University of the Ryukyus in 1994 and 1996, respectively. He is currently a senior associate professor in Information Systems Major, Faculty of International Studies, Meio University. His research interests include optimization and soft computing. He is a member of the Japanese Society of Evolutionary Computation.

**Takeo Yoshida** received the B.E. and M.E. degrees in electrical engineering from Nagaoka University of Technology and the D.E. degree in electrical engineering from Tokyo Metropolitan University in 1991, 1993 and 1997, respectively. He is currently an assistant professor in the Department of Engineering, University of the Ryukyus. His research interests include dependable computing, VLSI design, and graph theory. He is a member of IEEE and IPSJ.

**Morikazu Nakamura** received the B.E. and M.E degrees from University of the Ryukyus in 1989 and 1991, respectively, and D.E degree from Osaka University in 1996. He is currently a professor in Area of Computer Science and Intelligent Systems, Faculty of Engineering, University of the Ryukyus. His research interests include theory and applications on mathematical systems. He is a member of IEEE.