Proceedings of the World Congress on Engineering and Computer Science 2013 Vol I
WCECS 2013, 23-25 October, 2013, San Francisco, USA

Solving Constraint Satisfaction Problems by Artificial Bee Colony with Greedy Scouts*

Yuko Aratsu†, Kazunori Mizuno†, Hitoshi Sasaki†, Seiichi Nishihara‡

Abstract—**In this paper, we propose the artificial bee colony algorithm for solving large-scale and hard constraint satisfaction problems (CSPs). Our algorithm is based on the DisABC algorithm which is proposed to solve binary optimization. In our algorithm, two main improvements are adopted: (1) to supplement low local search ability of the ABC, a hybrid algorithm with greedy local search technique, called GSAT is combined and (2) in the scout bee phase, greedy scout bees are introduced, where bees construct new candidate solutions by using a partial assignment of the best solution probabilistically. We demonstrate that our algorithm can be effective for the hard instance which are concentrated in the phase transition and we also discuss that the search performance is varied by difference of the proportion of using partial assignments of the best solution.**

Keywords: constraint satisfaction, search, meta-heuristics, artificial bee colony, phase transition

1 Introduction

A constraint satisfaction problem (CSP) involves finding values for problem variables which are subject to given constraints specifying the acceptable combinations of values. Such combinatorial search problems are ubiquitous in artificial intelligence and pattern analysis, including scheduling, planning, resource allocation, or machine vision.

To solve large-scale and hard constraint satisfaction problems (CSPs) that are NP-complete, meta-heuristics for stochastic search approaches has been recently made remarkable progress[1]. Artificial bee colony (ABC) [2], based on the intelligent foraging behaviour of honeybee swarm, is one of typical meta-heuristics. Although it is expected that the ABC algorithm can be effective for many optimization problems, the ABC algorithm can not be directly applied to CSPs that are one of typical combinatorial search problems because the original ABC algorithm[2] has been applied to only continuous optimization problems. In contrast, Kashan, *et. al.* have proposed the ABC algorithm for binary optimization, which is an example of discrete or combinatorial optimization problems, called DisABC[3].

In this paper, we propose an ABC algorithm for solving hard and large-scale CSPs. The proposed algorithm is based on DisABC[3] modified to be applied to binary CSPs and a hybrid method combined with GSAT[4] to supplement local search ability. Furthermore, we provide greedy scout bees, which try to find a new food source, or candidate solution, by using source position information of the best solution so far. We experimentally demonstrate that our algorithm can efficiently solve hard binary CSPs around the phase transition region and discuss what types of behavior of greedy scout bees is more effective.

The CSP is well-known as an NP-complete problem, but actual problem instances with such computational complexity are found only in a locally limited region of the problem space. Recent studies have revealed that really hard problem instances tend to happen in situations very similar to physical phase transitions[5]. Problem instances within phase transition regions are very hard to solve for not only backtracking based systematic approaches but also repair-based stochastic approaches. Hence, it is important for the studies of meta-heuristics to place their interests on how well they cope with such hard problem instances within phase transition regions.

2 Problem definition and ABC algorithms

2.1 Constraint Satisfaction Problem

Let us briefly give some definition and terminology about CSPs. A CSP is defined as a triple (X, D, C) such that

- $X = \{x_1, \ldots, x_n\}$ is a finite set of variables,
- D is a function which maps every variable $x_i \in X$ to its domain D_i, i.e., the finite set of values that should be assigned to x_i, and

*This research was partially supported by Grant-in-Aid for Scientific Research (C), No. 25330267, Japan Society for the Promotion of Science, 2013-2016.

†Y. Aratsu, K. Mizuno and H. Sasaki are with the Department of Computer Science, Takushoku University, Hachioji, Tokyo 193-0985, Japan e-mail: yk.aratsu15@gmail.com, mizuno@cs.takushoku-u.ac.jp.

‡S. Nishihara is with Department of Computer Science, University of Tsukuba, Tsukuba, Ibaraki 305-8573, Japan.

ISBN: 978-988-19252-3-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

- C is a set of constraints, each of which is a relation between some variables which restricts the set of values that can be assigned simultaneously to these variables.

In this paper, let $D_1 = \cdots = D_n = D$ and $|D| = m$. We also employ binary CSPs which have only binary constraints, i.e., every constraint involves exactly two variables. Fig. 1 gives an example of binary CSP instances.

Binary CSP instances for using the experiments can be generated randomly. A class of randomly generated instances is characterized by 4 components, $\langle n,\ m,\ p_1,\ p_2\rangle$ where n is the number of variables, m is the domain size, or the number of values, p_1 is the probability of adding a constraint between two different variables, and p_2 is the probability of ruling out a pair of values between two constrained variables.

$X = \{x_1,\ x_2,\ x_3,\ x_4\}$
$D = \{a,\ b,\ c\}\ (= D_1 = \cdots = D_4)$
$C = \{c_{12},\ c_{23},\ c_{14}\}$
$c_{12} = \{(a\ b),\ (b\ c)\}$
$c_{23} = \{(c\ b),\ (b\ a),\ (b\ b)\}$
$c_{14} = \{(a\ c)\}$
Solutions: $(x_1\ x_2\ x_3\ x_4) = \{(a\ b\ a\ c),\ (a\ b\ b\ c)\}$

Figure 1: An example instance of binary CSPs.

2.2 ABC Algorithm

The ABC algorithm is a population-based metaheuristics inspired by the intelligent foraging behavior of honeybee swarm[2]. The foraging bees are classified into three categories of employed, onlookers, and scouts. Employed bees determine a food source within the neighbourhood of the food source in their memory. They share their information with onlookers within the hive and then the onlookers select one of the food sources. Onlookers select a food source within the neighbourhood of the food sources chosen by themselves. An employed bee of which the source has been abandoned becomes a scout and starts to search a new food source randomly. In the ABC algorithm, the position of a food source is a possible solution of the optimization problem and nectar amount of a food source corresponds to the fitness of an associated solution.

2.3 DisABC Algorithm

The ABC algorithm has been designed for optimization in continuous space and cannot work with vectors with discrete values. In contrast, the DisABC algorithm has been proposed to solve binary optimization[3]. In the DisABC algorithm, to measure distance between food sources, the "$-$" operator used in the original ABC algorithm is substituted with a dissimilarity measure of binary vectors by employing the Jaccard coefficient of similarity[3]. Letting $X_i = (x_{i1}, x_{i2}, \cdots, x_{id}, \cdots, x_{iD})$ and $X_j = (x_{j1}, x_{j2}, \cdots, x_{jd}, \cdots, x_{jD})$ where x_{id} and x_{id} can take only 0 or 1 values, $Dissimilarity\,(X_i, X_j)$ is defines as

$$Dissimilarity\,(X_i, X_j) = 1 - Similarity\,(X_i, X_j),$$

where

$$Similarity\,(X_i, X_j) = \frac{M_{11}}{M_{11} + M_{10} + M_{01}},$$

$$M_{11} = \sum_{d=1}^{D} I(x_{id} = x_{jd} = 1),$$

$$M_{10} = \sum_{d=1}^{D} I(x_{id} = 1, x_{jd} = 0),$$

$$M_{01} = \sum_{d=1}^{D} I(x_{id} = 0, x_{jd} = 1),$$

$$M_{00} = \sum_{d=1}^{D} I(x_{id} = x_{jd} = 0).$$

3 The Proposed Artificial Bee Colony Algorithm

3.1 Basic Strategies

In this paper, we propose an ABC algorithm for solving CSPs, summarized as follows:

i. The proposed algorithm is based on the DisABC algorithm which can solve binary optimization problems.

ii. CSP instances to be applied to the our algorithm are encoded or reexpressed to a binary optimization form.

iii. To supplement local search abilities of the ABC, GSAT proposed by Selman[4] is combined with our algorithm.

iv. Scout bees create candidate solution, i.e., new food source positions, by randomly using parts of the best solution so far.

Thus, our method can efficiently solve CSPs, although each CSP instance is needed to be reexpressed.

3.2 The Algorithm

Fig. 2 gives the proposed algorithm, in which DisABC is extended to solve binary CSPs. To apply binary CSP to our algorithm, CSP instances need to be reexpressed.

Fig. 4 gives the procedure to reexpress CSP instances. For example, when the CSP instance denoted in Fig. 1 is applied to this procedure, The binary version is output as shown in Fig. 5. In Fig. 2, the ith food source, $X_i^t (i = 1, \cdots, Ns)$, at the cycle t, which corresponds to the assignment of values to variables is first

ISBN: 978-988-19252-3-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

```
begin
  Initialization;
  For t = 1 to MaxCycle do
    EmployedBeesPhase();
    OnlookeredBeesPhase();
    If rand < Gp then
      GSAT(X^t);
    End if
    ScoutBeesPhase();
    Memorize the best solution achieved so far;
  End for
end

procedure EmployedBeesPhase()
  For i = 1 to Ns do
    Generate a new assignment V_i^t from X_i^t (and based on
    X_k^t (k ≠ i)) via NBSG proposed by [3];
    Evaluate the new solution;
    If Conf(V_i^t) < Conf(X_i^t) then
      X_i^t ← V_i^t;
      trial_i ← 0;
    Else
      Remember X_i^t;
      trial_i ← trial_i + 1;
    End if
  End for
end procedure

procedure OnlookerBeesPhase()
  For i = 1 to Ns do
    Calculate the probability proportional to the quality of
    food sources p_i;
    Produce a new assignment V_i^t from X_i^t (and based on
    X_k^t (k ≠ i)) selected depending on p_i via NBSG
    proposed by [3];
    Evaluate the new solution;
    If Conf(V_i^t) < Conf(X_i^t) then
      X_i^t ← V_i^t;
      trial_i ← 0;
    Else
      X_i^{t+1} ← X_i^t;
      trial_i ← trial_i + 1;
    End if
  End for
end procedure

procedure ScoutBeesPhase()
  If max{trial_i} ≥ limit then
    Replace the abandoned assignment with a new assignment
    by the procedure ConstructAssignmentByGreedyScout();
  End if
end procedure
```

Figure 2: Hyblrid DisABC

```
procedure ConstructAssignmentByGreedyScout()
  Initialize the abandoned assignment, X_i;
  k ← 0;
  While k < (deflection × n) do
    Select the j-th variable, X_ij, randomly;
    If X_ij is unassigned then
      X_ij ← best_ij;
      k ← k + 1;
    End if
  End while
  For j = 1 to n do
    If X_ij is unassigned then
      X_ij ← 0 or 1 (randomly);
    End If
  End for
end procedure
```

Figure 3: The proposed algorithm for Scout Bees.

generated at "Initialization". Then, the main three processes, EmployedBeesPhase, OnlookerBeesPhase, and ScoutBeesPhase, are repeated until a solution with no constraint violations, i.e., $Conf(X) = 0$, is found or t reaches $MaxCycle$. $Conf(X)$ is the number of constraint violations of X. GSAT, which can perform a greedy local search, is hybridized with our algorithm and is employed after the onlooker phase. Gp controls the rate of recalling the GSAT module in our algorithm.

In the scout bee phase of the original ABC algorithm, candidate solutions that cannot attain to a optimal solution are abandoned and scout bees attempt to generate new candidate ones randomly. This operation seems to be effective in the sense of preventing candidate solutions from getting stuck in locally optimal solutions. However, it can make the search more slow to conduct completely random generation of new candidate solutions and possibly fair partial solutions may exist in the abandoned solutions. We improve the scout bee phase, as shown in Fig. 3, where a partial assignment which is equal to or less than a certain proportion, say $deflection$, of the whole assignment is generated by assigning that of the best solution so far and the remaining assignment is generated randomly when a scout bee generates a new candidate solutions.

4 Experiments

4.1 Experimental Settings

To evaluate the efficiency of the proposed methods, we attempt to briefly conduct the experiments. We employ two types of randomly generated binary CSP instances: $\langle 30, 4, 0.14, p_2 \rangle$ for 19 cases of $p_2 = 0.30$, 0.32, 0.34, 0.36, 0.38, 0.40, 0.42, 0.44, 0.46, 0.48, 0.50, 0.52, 0.54, 0.56, 0.58, 0.60, 0.62, 0.64 and 0.66, and $\langle 50, 4, 0.14, p_2 \rangle$ for 19 cases of $p_2 = 0.14$, 0.16, 0.18, 0.20, 0.22, 0.24, 0.26, 0.28, 0.30, 0.32, 0.34, 0.36, 0.38, 0.40, 0.42, 0.44, 0.46, 0.48

ISBN: 978-988-19252-3-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

```
begin
  Generate n × m variables;
  For i = 1 to n do
    For j = (i + 1) to n do
      If c_ij ∈ C then
        For k = 0 to (m − 1) do
          If k == (c_i −′ a′) then
            c_i('a'+k) ← 1;
          Else
            c_i('a'+k) ← 0;
          End if
          If k == (c_j −′ a′) then
            c_j('a'+k) ← 1;
          Else
            c_j('a'+k) ← 0;
          End if
        End for
      End if
    End for
  End for
end
```

Figure 4: The procedure to reexpress CSPs.

$X = \{(x_{1a},\ x_{1b},\ x_{1c}),$
$\quad (x_{2a},\ x_{2b},\ x_{2c}),$
$\quad (x_{3a},\ x_{3b},\ x_{3c}),$
$\quad (x_{4a},\ x_{4b},\ x_{4c})\}$
$D = \{0,\ 1\}\ (= D_{1a} = \cdots = D_{4d})$
$C = \{c_{1*2*},\ c_{2*3*},\ c_{1*4*}\}$ (*: a, b, or c)
$\quad c_{1*2*} = \{(1\ 0\ 0\ 0\ 1\ 0),\ (0\ 1\ 0\ 0\ 0\ 1)\}$
$\quad c_{2*3*} = \{(0\ 0\ 1\ 0\ 1\ 0),\ (0\ 1\ 0\ 1\ 0\ 0),\ (0\ 1\ 0\ 0\ 1\ 0)\}$
$\quad c_{1*4*} = \{(1\ 0\ 0\ 0\ 0\ 1)\}$
Solutions: $(x_{1*}\ x_{2*}\ x_{3*}\ x_{4*}) = \{(1\ 0\ 0\ 0\ 1\ 0\ 1\ 0\ 0\ 0\ 0\ 1),$
$\quad (1\ 0\ 0\ 0\ 1\ 0\ 0\ 1\ 0\ 0\ 0\ 1)\}$

Figure 5: An example instance of the reexpressed binary CSPs.

and 0.50 we randomly generate 100 instances per case, i.e., a total of 1900 × 2 generated instances, whose search space size is $4^{30}(\simeq 10^{18})$ and $4^{50}(\simeq 10^{30})$, respectively. However, the search space size of reexpressed ones is 2^{120} $(\simeq 10^{36})$ and 2^{200} $(\simeq 10^{60})$. These instances are located around phase transition[5] regions derived by the following equation[6]:

$$\kappa = \frac{n-1}{2} p_1 \log_m(\frac{1}{1-p_2}). \quad (1)$$

In particular, $\kappa = 1.015(\simeq 1)$ in the case of $p_2 = 0.50$, corresponding to critically constrained regions.

Let us clarify the parameter settings of the methods. The number, Ns, of food sources is set to 50, '$MaxCycle$' denoted in Fig. 2 is set to 10,000, and the number, $limit$, of trials for releasing a food source is set to $n \times m$. As for parameters on GSAT in our algorithm, G_p and $MAX\text{-}FLIPS$ are set to 0.05 and 10, respectively. The parameter, $deflection$, which is the proportion making the partial assignment inherit from the best solution in the scout phase, is set to 0.1, 0.3, 0.5, 0.7, and 0.9[1]

We also briefly conduct experiments on naive GSAT, in which parameters, $MAX\text{-}TRIES$ and $MAX\text{-}FLIPS$ in [4] are set to 5,000 and 100, respectively.

We evaluate the percentage of solved CSP instances, that is, the proportion of the number of instances for which the method can find a solution to all instances to be tried to search and average cycles required for solving. The method is implemented in language Java on a PC with 3.07GHz of Intel Core i7 880 and 4GBytes of RAMs.

4.2 Experimental Results and Discussion

Figs. 6, 7, 8, and 9 give the results. Figs. 6 and 8 show the percentage of solved CSP instances. Figs. 7 and 9 show the average of total cycles in our approach.

As shown in Figs. 6 and 7, the percentage of solved CSPs is the lowest around $p_2 = 0.42$ for every value of the parameter $deflection$. More search cycles are required around $p_2 = 0.46$. Thus, hard problem instances are concentrated in the region from $p_2 = 0.42$ to $p_2 = 0.46$ for the 30 variables case. As for each $deflection$ value, there is few difference in the percentage of solved instances for $deflection = 0.0 \sim 0.5$. The search cycles can also be held down in those parameters. However, the lower percentage appears for $deflection = 0.7$ and 0.9. In particular, in $deflection = 0.9$, both of the percentage and search cycles become substantially worse.

On the other hand, as shown in Figs. 8 and 9, the hard problem instances are concentrated in the region from $p_2 = 0.30$ to $p_2 = 0.36$. As well as the results of the 30 variables case, the search can be more effective and almost synchronized for $deflection = 0.5$ or less. When $deflection = 0.5$, the percentage of solved instances is the highest in $p_2 = 0.30$ where the hardest instances seems to be concentrated.

These results demonstrate that the search performance can grow down by using partial assignments with high probability. This seems to be the reason why the candidate solutions being inclined to get stuck in locally optimal solutions can be easily constructed by inheriting more partial assignments from the best solution. In contrast, the search performance can slightly be more effective for the lower proportion, e.g., $deflection = 0.3$ and 0.5 than for $deflection = 0.0$, i.e., scout bees in the original ABC algorithm. Thus, our greedy scout bees can make the search more efficiently.

[1] In $deflection = 1$, new assignments generated in the scout phase are equal to the best solution. In $deflection = 0$, those assignments are generated completely randomly.

ISBN: 978-988-19252-3-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

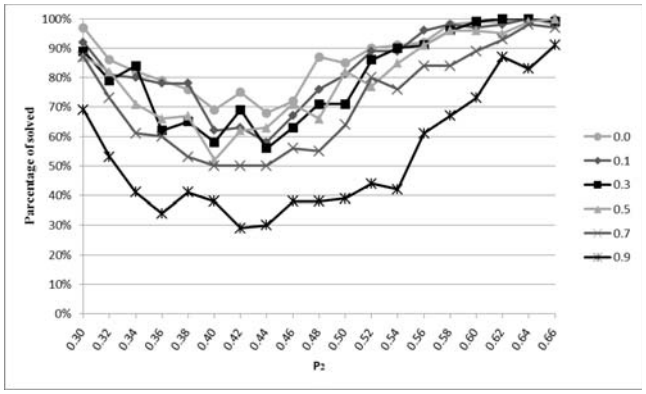

Figure 6: Experimental result of the 30 variables case on the percentage of success.

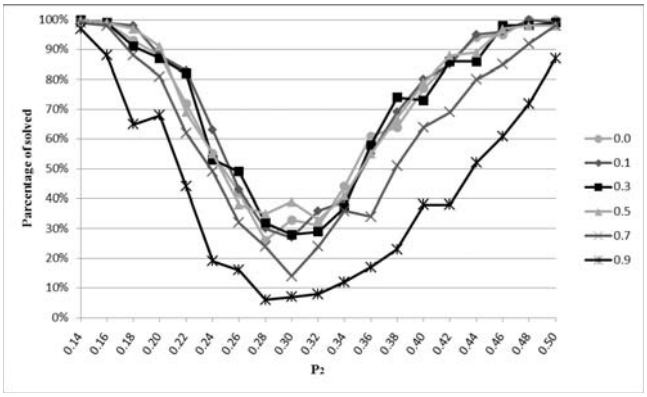

Figure 8: Experimental result of the 50 variables case on the percentage of success.

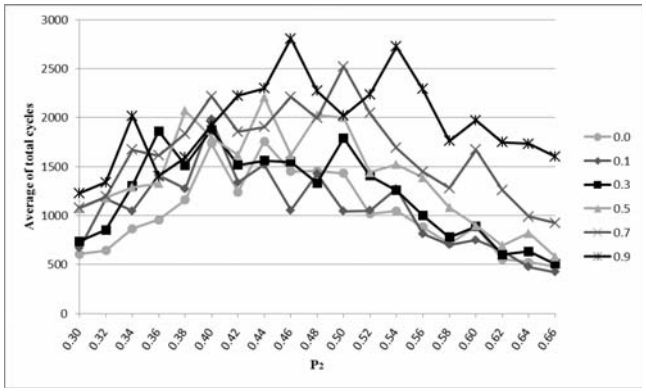

Figure 7: Experimental result of the 30 variables case on average of total cycles.

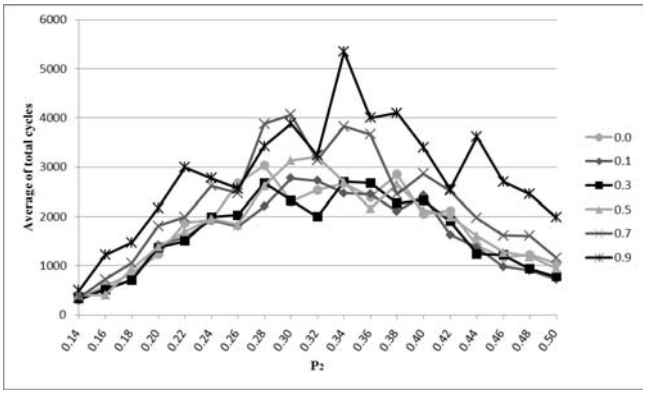

Figure 9: Experimental result of the 50 variables case on average of total cycles.

5 Conclusion

We have described a new approach to deal with hard and large-scale CSPs using the ABC algorithm in this paper. In our method, DisABC[3] is modified to be applied to binary CSPs and a hybrid method combined with GSAT[4] to supplement local search ability. Moreover, we provide the parameter which is the proportion making the partial assignment inherit from the best solution in the scout phase to find a new food source, or candidate solution, by using source position information of the best solution so far.

We conduct brief experiments, demonstrating that our algorithm can be effective to solve CSP instances. Our most important future works should consist in immediately conducting more experiments and comparing with other meta-heuristics and swarm intelligence approaches such as particle swarm optimization[7].

References

[1] Mizuno, K., Nishihara, S., *et. al.*: "Population migration: a meta-heuristics for stochastic approaches to constraint satisfaction problems", Informatica, Vol.25, No.3, pp.421-429 (2001).

[2] Karaboga, D., Basturk, B.: "On the performance of artificial bee colony (ABC) algorithm", Applied Soft Computing 8 pp.687-697 (2008).

[3] Kashan, M.H., Nahavandi, N. and Kashan, A.H.: "DisABC: A new artificial bee colony algorithm for binary optimization", Applied Soft Computing 12, pp. 342–352 (2012).

[4] Selman, B., Levesque, H., and Mitchell, D.: "A New Method for Solving Hard Satisfiability Problems", Proceedings of AAAI' 92, pp. 440-446 (1992).

[5] Hogg, T., Huberman, B. A., Williams, C. P.: "Phase transition and search problem", Artificial Intelligence, Vol. 81, pp. 1-16 (1996).

ISBN: 978-988-19252-3-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)

[6] Clark, D. A., Frank, J., Gent, I. P., MacIntyre, E., Tomov, N., Walsh, T.: "Local Search and the Number of Solutions", Proc. CP'96, pp. 119-133 (1996).

[7] Schoofs, L., Naudts, B.: "Swarm intelligence on the binary constraint satisfaction problem", Proceedings of the IEEE Congress on Evolutionary Computation (CEC), pp.1444-1449 (2002).

[8] Aratsu, Y., Mizuno, K. *et. al.*: "Artificial Bee Colony for Constraint Satisfaction Problems", SCIS-ISIS2012, pp. 2283–2286 (2012).

ISBN: 978-988-19252-3-7
ISSN: 2078-0958 (Print); ISSN: 2078-0966 (Online)