Particle Swarm in Binary CSPs with Dynamic Variable Ordering

Qingyun Yang [††], Jigui Sun [†††], and Juyang Zhang [††]

[†]College of Computer Science and Technology, Jilin University, Changchun 130012 China
[††]Key Laboratory for Symbolic Computation and Knowledge Engineering of Ministry of Education, Jilin University, Changchun 130012 China
[†††]Open Laboratory for Intelligence Information Processing, Fudan University, Shanghai 200433 China

Summary
The variable ordering of constraint satisfaction problems affect the performance of search algorithms in CSPs. Dynamic Variable Ordering (DVO) has more advantage in improving the performance of search algorithms than static variable ordering. It is a newly developed method recent years that using particle swarm algorithm to solve binary constraint satisfaction problems, which is a global stochastic optimized algorithm making use of swarm to search the whole solution space, and each particle represents a candidate solution of the problem. The algorithm discovers a solution satisfying condition specified of the solution space by acting each other among these particles. We add the dynamic variable ordering to the particle swarm algorithm in constraint satisfaction problems by improving the evaluation function of the particle swarm algorithm, which enhances the searching efficiency of particle swarm algorithm in CSPs and finds the solution of CSPs faster. Kinds of random constraint satisfaction problem experiments indicated that our efforts were effective.

Key words:
Discrete Particle Swarm, Constraint Satisfaction Problem, Dynamic Variable Ordering

1. Introduction

Constraint Satisfaction Problems (CSPs) are the most exciting research area of artificial intelligence over last decades. In real life, many problems such as timetabling, resource allocation, planning, configuration, vision etc. can be modeled as some certain type CSPs. It is a powerful tool to model and to solve many kinds of combinatorial problems, which has attracted widespread commercial interest as well. Most of those combinatorial problems are NP-Complete in general [1]. Researching into solving CSPs has been lasted for a long time and different directions of the subject have been approached: finding the CSPs' solutions from the possible solution space ([2],[3],[4]), reducing a CSP to a simpler or equivalent problem ([5],[6],[7],[8].[9]), and synthesizing the CSPs' solutions from partial solutions ([10],[11].[12]). But there are no polynomial algorithms to solve CSP so far. The dominating idea is backtracking that explores the search space in a systematic and complete way. Backtracking finds values for variables subject to a set of constraints. It improves the performance of the simple depth-first search by cutting down the search space [2]. Many surveys on backtrack algorithms can be found in [13].

The desirable features of system and completeness in backtrack search become intractable on hard combinatorial problems for the exponential time complexity [14,15,16]. This is particularly true when the problem is huge and system reduction cannot reduce the problem enough to make complete search feasible. Hence, incomplete search approaches have been proposed that leave out combinatorial explosion.

Particle Swarm Optimization (PSO) is an efficient stochastic global search optimization technique [17], which makes use of a particle population to search the whole solution space. Each particle represents a candidate solution of the problem being solved. The particle swarm algorithms find optimal regions of complex search space through the interaction of individual particle in the population. [18] proposed a discrete version of particle swarm algorithm to solve binary CSPs (we call it PS-CSP), which redefined the equations presented in basic particle swarm algorithm but did not consider the characteristic of CSPs themselves.

The remainder of the paper is organized as follows. Section 2 presents Constraint Satisfaction Problem. We also provide an overview of PS-CSP. In section 3 we propose a hybrid algorithm of particle swarm solving binary CSPs with Dynamic Variable Ordering (DVO) [12], in which DVO improves the evaluation function on selecting more promising particle as the best particle for the next iteration. An experimental evaluation on randomly generated constraint satisfaction problems is given in section 5 and some remarks conclude this paper.

2. Preliminaries

Manuscript revised January 2006.

A constraint network or constraint satisfaction problem (CSP) is a triplet (X, D, C), here X = {x_1, x_2, … x_n} is a set of Variables, which may take on values from a set of domains D = {D_1, D_2, … D_n}, and a set of constraints C = {C_1, C_2, … C_m}. Each constraint C_i is a pair (S_i, R_i), where R_i is relation $R_i \subseteq D_{s_i}$, and $D_{s_i} = \bowtie_{x_j \in s_i} D_j$, defined on a subset of variables $S_i \subseteq X$ called the scope of C_i. The relation denotes all compatible tuples of D_{s_i} permitted by the constraint. A binary constraint C_{ij} on variable X_i and X_j is a set of pairs too, C_{ij} allows for X_i to take the value v_i and X_j to take the value v_j iff $(v_i, v_j) \in C_{ij}$ and we say the binary constraint is satisfied otherwise it is violated. We call a CSP binary constraint satisfaction problem iff all the constraints in CSP are binary constraints. A solution of a constraint satisfaction problem is an assignment of values to the set of X such that all the constraints are satisfied simultaneously. Conflicting number of x is the number violated in constraints with the related variable x, $conflict(x) = \sum_{i=1}^{|C|} violate(c_i)$.

PS-CSP is a discrete particle swarm algorithm based on the idea of [19,20], which redefined the operators for velocity. PS-CSP throws all the particles into the problem space just the same as the basic particle swarm algorithm does at the initial phase. Velocity is a real number in basic particle swarm algorithm while the subtraction of two positions resulting in a velocity in PS-CSP. Supposing $\vec{x}$ and $\vec{y}$ are positions. Then $\vec{v} = \vec{x} - \vec{y} = \{\vec{y} \to \vec{x}\}$ denotes a velocity. The long arrow indicates the change of position. Additional operators need to be redefined for the changing of velocity. The addition of a position with a velocity results in a position, $\vec{x} \oplus \vec{v}$, where suppose $\vec{x}$ is a position and $\vec{v} = \vec{z} - \vec{y}$. Then $\vec{x} \oplus \vec{v}$ equals the position produced in by $\vec{x} \oplus \vec{v} = \begin{cases} z_i & if \quad x_i = y_i \\ x_i & otherwise \end{cases}$. The next operator is a velocity add another velocity resulting in a new velocity $\vec{x} \oplus \vec{v}$, where $\vec{v} = \vec{b} - \vec{a}$, $\vec{w} = \vec{y} - \vec{x}$. Then the newly created velocity is $\vec{v} \circ \vec{w} = \begin{cases} a_i \to y_i & if \quad b_i = x_i \\ a_i \to b_i & otherwise \end{cases}$. The last operation is the multiplication of a velocity and a coefficient, according to a corresponding position. The multiplication results in a velocity vector $\varphi \otimes \vec{v}$, where $\vec{v} = \vec{y} - \vec{x}$. The result velocity vector produced in by $\varphi \otimes \vec{v} = \begin{cases} x_i \to x_i & if \quad nbconf_i \le \varphi \\ x_i \to y_i & otherwise \end{cases}$, $nbconf_i$ is the conflicting number of position $\vec{x}$ with offset i. The calculation of conflicting number uses the equation defined above. At last the discrete particle swarm algorithm can be rewritten as follows:

$$\vec{v}^{\,t} = (\varphi_1 \otimes (\vec{p} - \vec{x}^{\,t-1})) \circ (\varphi_2 \otimes (\vec{g} - \vec{x}^{\,t-1}))$$

$$\vec{x}^{\,t} = \vec{x}^{\,t-1} \oplus \vec{v}^{\,t-1}$$

and the algorithm's pseudocode is as follows:

```
random initialization of the swarm
set localbest and globalbest
while stopping criterion not satisfied
do
    for i = 0 to swarmSize – 1 do
       for j=0 to n-1 do
         nbconf←number of conflicts for
x_ij^t-1 in particle i
       if nbconf  > φ1 then
            v' ← bestSoFar_ij − x_ij^t-1
       else v' ← x_ij^t-1 − x_ij^t-1
       end if
       if nbconf  > φ2 then
           if deflection then
                v'' ← Rand − x_ij^t-1
           else v'' ← globalBest_j − x_ij^t-1
           end if
           else v'' ← x_ij^t-1 − x_ij^t-1
           end if
           x_ij^t ← x_ij^t-1 ⊕ (v' ∘ v'')
       end for
       calculate the evaluation and
determine the local best particle
    end for
    determine global best particle
end while
```

3. Particle swarm in binary CSPs with dynamic variable ordering

In the dynamic variable ordering algorithm based on back-tracking, the next variable to be instantiated is to find the one who has the minimum domain among all the uninstantiated variable set. Then give a value within the domain to the selected variable. After the instantiation of the minimum domain variable, checking consistency can be executed for first fail principle. Checking consistency can remove redundant inconsistent values from domains to reduce domains of variables further. Until all the variables instantiated, the algorithm determines whether a solution found or not.

All variables instantiated in particle swarm algorithm, and the particles in swarm are composed of these instantiated values. It is a key to the question how to confirm the size of domain of a certain variable. We treat the variable to be counted domain as a uninstantiated one. Then we leave the values to satisfy the constraints related to the selecting variable as its domain. We define $\mathrm{d}(v) = k - conflict(v)$ to hold the selecting variable's domain. Where k is the number of constraints the selecting variable relates to, d(v) is the size of domain of variable v in the dynamic variable ordering. Use the method above to calculate all variables in constraint network. We can get the minimum domain variable in the network.

Particle swarm algorithm needs to find the global best particle to influence the present position of particles and to guide the direction of the swarm. The algorithm judges the goodness of particles by evaluation function. The particle with the minimum evaluation is chosen from the population and treated as the global best particle in PS-CSP. Less evaluation means more constraints satisfied, and more close to the solution of the problem. We still make use of conflicting number to evaluate the goodness of particles, but use dynamic variable ordering to rewrite the evaluation function as follows:

$$fitness(x_i) = \sum_{j=1}^{n} (d(v_j) + 2)^{nbconf_{v_j}} - n$$

x_i is the ith particle in the swarm, n is the number of variable in the constraint network, $d(v_j)$ denotes the size of domain of variable v_j under dynamic variable ordering. $nbConf_{v_j}$ means the conflicting number of variable v_j.

After redefining the evaluation function, particle swarm algorithm to solve binary CSPs can be modified as follow (we call it DVO-PS-CSP):

```
random initialization of the swarm
set localbest and globalbest
while stopping criterion not satisfied
do
  for i = 0 to swarmSize – 1do
    for j=0 to n-1 do
      update particle velocity and
position
    end for
    compute variables domain
```

$$fitness(x_i) = \sum_{j=1}^{n} (d(v_j) + 2)^{nbconf_{v_j}} - n$$

```
      if fitness_i < fitness_bestSoFar then
```

$$bestSoFar_i \leftarrow x_i$$

```
      end if
      if fitness_i unchanged for noHope
times then random re-initialization of
x_i
      end if
  end for
```

determine golbalBest

4. Experimental Results

We compared these algorithms with the classes of randomly generated constraint network [21]. Four parameters were taken into account: n the number of variables, d the number of values per variable, pc the probability that a constraint R_{ij} between two variables exists, and pu the probability in existing relations R_{ij} that a pair of values $R_{ij}(a, b)$ is allowed. The result given for each class is the average for ten instances of problems in the class so as to be more representative of the class.

Particle Swarm is a stochastic searching algorithm. We test each instance with 100 times for impartial results and treat the average of 100 multiple 10 as the final result. To solve a CSP instance is to label each variable a value that all constraints are satisfied, thus for showing the performance of Particle Swarm, we guarantee the randomly generated CSP instances are all solvable by performing with systematic algorithm of backtracking. In all experiments, we generate constraint graph with 15 variables having 15 possible values, and swarm size of 50 is used. As soon as there is a particle satisfying all constraints in the constraint network simultaneously then the algorithm terminates. If a certain particle reaches the maximal iteration while no solution is found then it reinitializes the particle named no hope system. The

coefficient deflection is 2/n (n is the number of variables in the constraint network).

pc keeps fix in Fig. 1 and Fig. 2, pu increases progressively, larger pu means more candidate value pair satisfying constraints in constraint network. pu=1 indicates all constraints can be satisfied by all value pair. Thus larger pu results in higher probability to find the solution for higher possibility the instantiated values of the two variables in a constraint satisfying the constraint. From Fig. 1 and Fig. 2, we can see that both algorithm decrease in the iteration when pu increase, but DVO-PS-CSP is more efficient than PS-CSP on iteration and time.

pc increases progressively in Fig. 3, that is to say that increases the number of constraints in the constraint network. In order to guarantee each problem can be solved, value pairs satisfied constraints increase too while increasing the number of constraints. We can see in the Fig. 3 that after using the dynamic variable ordering, the algorithm reduces the iteration greatly. Only one problem instance's iteration is close with both algorithms. Contrasting runtime with PS-CSP in Fig. 4 we can see that the improved algorithm run less time than PS-CSP. But it does not have the same effectiveness as the iteration. This is because when calculate the evaluation of the particle, more complicated than the original one. PS-CSP only needs to count the number of conflict, and DVO-PS-CSP not only needs to calculate the conflicting number, but also needs to compute the number of constraints satisfied that each variable related to. Although the computation of evaluation is slightly long, this complicated calculation is undoubtedly worthy on the average result.

5.Conclusion

Backtracking is an ordinary way in solving constraint satisfaction problems, which selects an unlabelled variable to instantiate each time. When the instantiated variable doesn't meet with the constraints, the algorithm backtracks to a suitable search point. Backtracking algorithm traverses the entire solution space. The system and completeness in backtrack search become intractable on hard combinatorial problems for the exponential time complexity. This is especially true when the problem is huge and system reduction cannot reduce the problem enough to make complete search feasible. The priority order that the variable searches for can influence the efficiency of the algorithm, in the search of backtracking, the proper variable ordering can reduce the search space. Particle Swarm algorithm is a stochastic global search technique, which makes use of a particle population to search the whole solution space. Each particle represents a candidate solution of the problem being solved. The particle swarm algorithms find optimal regions of complex search space through the interaction of individual particle in the population. Each particle is made up of all the instantiated values in the constraint network. We proposed a hybrid algorithm combining particle swarm algorithm with dynamic variable ordering. The hybrid algorithm makes it search the solution space by variable ordering so as to expect it can find the solution faster. Randomly generated problem instances denote the hybrid algorithm has stronger search ability, whose search efficiency improves averagely at double.

Acknowledgment

This work was supported in part by the National Natural Science Foundation of China under grant no. 60273080 and no. 60473003. This work also was supported in part by the Outstanding Youth Foundation of Jilin province of China under grant no. 20030107.

References

[1] M. R. Garey and D. S. Johnson. Computers and Intractability. A Guide to the Theory of NP-Completeness. W. H. Freeman and Company, 1979.

[2] E.C. Freuder. A sufficient condition for backtrack-free search. J. ACM, 1982, 29(1): 24-32.

[3] E.C. Freuder. A sufficient condition for backtrack-bounded search. J.ACM, 1985, 32(4): 755-761.

[4] R. Dechter, J. Pearl. Tree clustering for constraint networks. Artificial Intelligence, 1989, 38(3) 353-366.

[5] A. K. Mackworth, E. C. Freuder. The complexity of some polynomial consistency algorithms for constraint satisfaction problems. Artificial Intelligence, 1985, 25: 65-74.

[6] Romuald Debruyne, Christian Bessière. Domain Filtering Consistencies. Journal of Artificial Intelligence Research, 2001, 14: 205-230.

[7] Christian Bessière, Jean-Charles Régin. Refining the Basic Constraint Propagation Algorithm. IJCAI 2001. 309-315.

[8] C.Han, C.Lee. Comments on Mohr and Henderson's path consistency algorithm. Artificial Intelligence, 1988, 36: 125-130.

[9] R.Morh, T.Henderson. Arc and path consistency revisited. Artificial Intelligence, 1986, 28: 225-233.

[10]E.Freuder. Synthesizing constraint expressions. Communications of the ACM, 1978, 21(11): 958-966.

[11] Wanlin Pang, Scott D. Goodwin: A Graph Based Synthesis Algorithm for Solving CSPs. FLAIRS Conference 2003. 197-201.

[12] E.Tsang. Foundations of constraint satisfaction. Academic Press, San Diego, CA, 1993.

[13] R. Dechter, D. Frost. Backtracking algorithms for constraint satisfaction problems; a survey. in Constraints, International Journal, 1998.

[14] J.R.Bitner, E.Reingold. Backtrack programming techniques. Communications of the ACM, 1975, 18(11): 651-656.

[15] G. Kondrak, Peter van Beek. A Theoretical Evaluation of Selected Backtracking Algorithms. Artificial Intelligence Journal, 1997, 89(1-2): 365-387.

[16] R. Dechter, D. Frost. Backtracking algorithms for constraint satisfaction problems. Technical report, University of California, Irvine, 1999.

[17] J. Kennedy, R. C. Eberhart. Particle Swarm Optimization. In: IEEE International Conference on Neural Networks, Perch, Australia. 1995. 1942-1948.
[18] L. Schoofs, B. Naudts. Swarm intelligence on the binary constraint satisfaction problem. In: Proceedings of the IEEE Congress on Evolutionary Computation (CEC 2002), Honolulu, Hawaii USA. 2002. 1444-1449.
[19] J. Kennedy and R. C. Eberhart. A discrete binary version of the particle swarm algorithm. In International Conference on Systems. Man, and Cybernetics, 1997.
[20] M. Clerc. Discrete particle swarm optimization: A fuzzy combinatiorial black box. In http://clerc.maurice.free.fr/PSO/, 2000.
[21] C. Bessière. Arc-consistency and arc-consistency again. Artificial Intelligence 65(1994): 179-190.

Qingyun Yang received the B.E. degree from Jiangxi Univ. of Science and Technology in 2000, and received M.Sc degree from Jilin Univ. in 2003. He studies as Ph. D candidate since 2003. His research interest includes constraint satisfaction problem, swarm intelligence, scheduling, artificial life.

Jigui Sun received the B.E., M.Sc and Dr. degrees from .Jilin Univ. in 1987, 1990 and 1993, respectively. He has been a professor (from 1997) and Ph.D superior in the JiLin Univ. since 2000. His research interest includes automation reasoning, constraint programming, decision support system, GIS, computation intelligence.

Juyang Zhang received the B.E. degree from East China Institute of Technology in 2000, and received M.Sc degree from Jilin Univ. in 2003. He studies as Ph.D candidate since 2003. His research interest includes constraint based scheduling, constraint satisfaction problem, constraint logic programming.

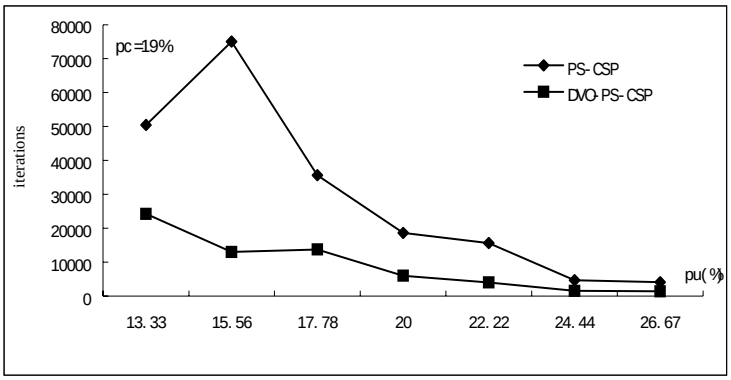

Fig. 1. pc=19%, pu increases.Comparison of iteration

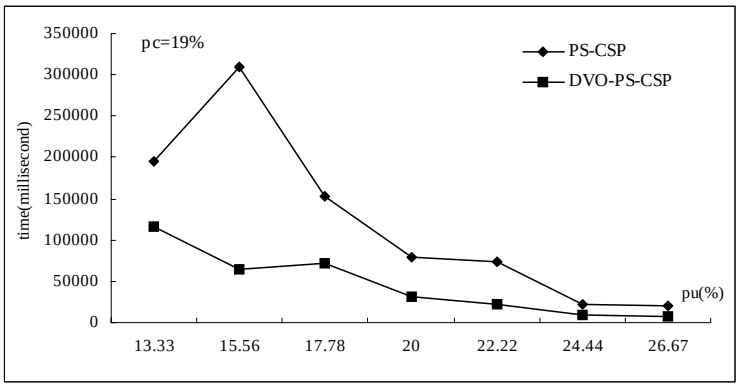

Fig. 2. pc=19%, pu increases.Comparison of runtime

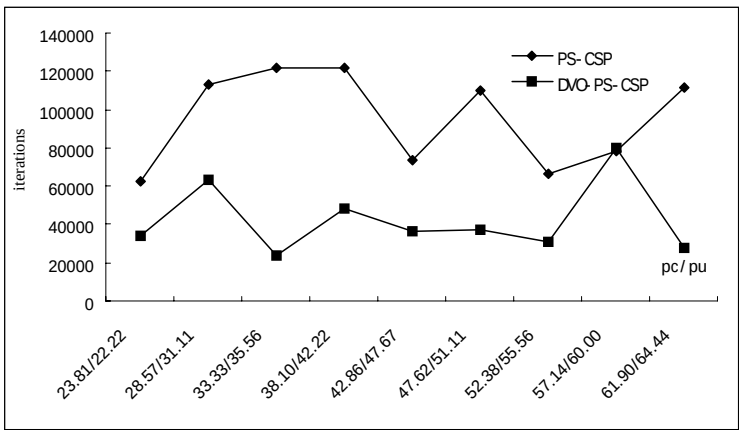

Fig. 3. Both pc and pu increase.Comparison of iteration

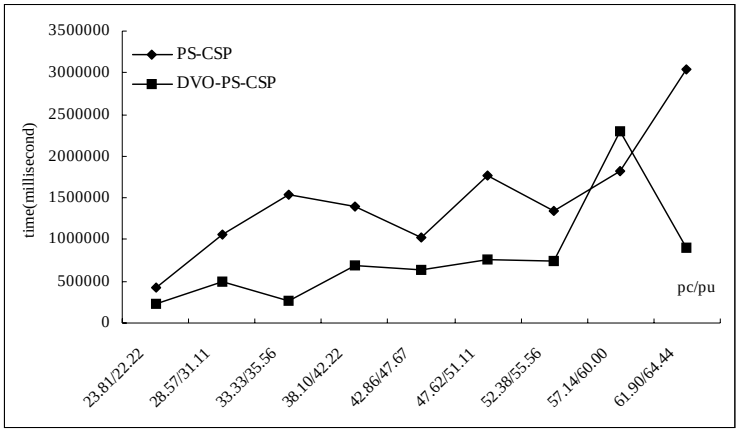

Fig. 4. Both pc and pu increase.Comparison of runtime