A Hybrid Approach Using Particle Swarm Optimization and Parallel Simulated Annealing: A Case Study of n-Queens Problem

Pourya Jafarzadeh and Vahid Mohammadi Saffarzadeh

***Abstract*—This paper presents a hybrid approach by a combination of particle swarm optimization (PSO) and parallel simulated annealing (PSA) algorithms. PSO is a population based heuristic method that sometimes traps in local maximum. To cope with this problem, we utilized simulated annealing. However, since SA is extremely greedy regarding the number of iterations, a parallel approach can be performed to decrease the total iterations. In this article, we utilized discrete PSO to achieve a good local maximum. Then parallel SA (PSA) is employed to escape from this locality. Study on the n-queens problem shows that PSO-PSA is promising in solving constraint satisfaction problems.**

***Index Terms*—Constraint satisfaction, parallel simulated annealing, particle swarm optimization.**

I. Introduction

A constraint satisfaction (CSP) is defined by a set of variables and a set of constraints. Variable has a nonempty domain of possible values. Each constraint involves some subset of the variables and specifies the allowable combination of values for the subset. A state of the problem is defined by an assignment of values to some or all variables. A complete assignment is one in which every variable is mentioned, and a solution to CSP is a complete assignment that satisfies all the constraints. Some CSP problems also require a solution that maximizes an objective function [1].

The n-queens problem is a CSP that consists of placing n queens on an $N \times N$ chess board, so that they do not attack each other, i.e. on every row, column or diagonal, there is only one queen. Its complexity is $O(n!)$ [2], [3]. There are many heuristics for solving n-queen problems, some of these heuristics cooperate better with some search methods than the others. The complexity of heuristic used in this paper is $O(n)$. There are also several search strategies for n-queens problem such as Depth First Search, Beam Search, Branch and Bound, local search methods and Evolutionary algorithms (EA).

Particle Swarm Optimization (PSO) and Simulated Algorithm (SA) are well known heuristic methods that both are inspired from real phenomenon. There are many optimization problems, which have been solved with these two algorithms.

The sections of the paper are as follows: Section II reviews the basic and discrete forms of PSO. The Section III and Section IV review the SA algorithm and its parallel approach, respectively. In Section V we described our hybrid algorithm, and Section VI summarizes experimental results. Finally, conclusions and future works are presented in Section VII.

Manuscript received November 8, 2013; revised February 18, 2014.

The authors were with Shahid Chamran University of Ahvaz, Iran (e-mail: pouryajafarzadeh@gmail.com, vahid.msaffarzadeh@gmail.com).

II. Particle Swarm Optimization

A. Basic PSO

The Particle Swarm Optimization (PSO) [Kennedy and Eberhart, 1995, Eberhart *et al.*, 2001] evolved from an analogy drawn with the collective behavior of the animal displacements [4]. The system is initialized with a population of random solutions and searches for optima by updating potential solution over generation. In PSO, the potential solutions, called particles, "fly" through the problem space by following the current better performing particle [2]. The solution updating can be represented by the concept of velocity [5]. By definition, a velocity is a vector or, more precisely, an operator, which, applied to a position (solution), will give another position. It is in fact a displacement, called velocity because the time increment of the iteration is always implicitly regarded as equal to 1 [6]. Velocity of each particle can be modified (updated) by the following equation:

$$v_i^{k+1} = w.v_i^k + c_1.rand_1 \times (pbest_i - s_i^k) + c_2.rand_2 \times (nbest_i - s_i^k) \quad (1)$$

where v_i^k is velocity of particle i at iteration k, w is weighting function, c_j is weighting coefficients, *rand* is random number between 0 and 1, s_i^k is the current position of particle i at iteration k, $pbest_i$ is the best state of particle i, and $nbest_i$ is the best state among the neighbors of particle i (until iteration k) [5]. A general flowchart of basic PSO is shown in Fig. 1.

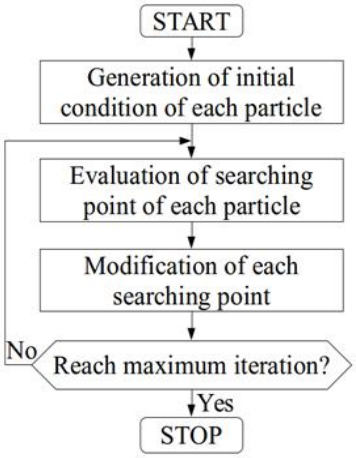

Fig. 1. A general flowchart of PSO [5].

DOI: 10.7763/IJCEE.2014.V6.828

B. Discrete PSO

The basic PSO treats nonlinear optimization problem with continuous variables. However, practical engineering problems are often formulated as combinatorial optimization problems. Kennedy and Eberhart developed a discrete binary version of PSO for these problems. They proposed a model wherein the probability of an agent's (particle) deciding yes or no, true or false and 0 or 1 by the following factors:

$$P(s_i^{k+1}=1)=f(s_i^k, v_i^k, pbest_i, nbest_i) \quad (2)$$

The parameter *v*, a particle's tendency to make one or the other choice, will determine a probability threshold. If *v* is high, the particle is more likely to choose 1, and lower values favor zero choice. Such a threshold requires staying in the range [0, 1]. The proper function for this feature is the sigmoid function:

$$sig(v_i^k)=1/(1+\exp(-v_i^k)) \quad (3)$$

Like the basic continuous version, updating formula for the binary (discrete) version of PSO can be described as follows [5]:

$$\begin{aligned} v_i^{k+1} &= v_i^k + rand_1 \times (pbest_i - s_i^k) + rand_2 \\ &\quad \times (nbest_i - s_i^k) \\ if\ \ \rho &< sig(v_i^{k+1}) \quad then \quad s_i^{k+1}=1; \\ else \quad s_i^{k+1} &= 0; \end{aligned} \quad (4)$$

where *rand* and ρ are random numbers in [0,1]. The entire algorithm of the binary version of PSO is almost the same as that of the basic continuous version (Fig. 1) except for the above equations [5].

III. Simulated Annealing (SA)

In statistical mechanics, a physical process called annealing is often performs in order to relax the system to a state with minimum free energy [7]. The idea to use annealing technique in order to deal with optimization problems gave rise to the simulated annealing technique. It consists in introducing a control parameter in optimization, which plays the role of temperature. The "temperature" of the system to be optimized most have the same effect as the temperature of the physical system: it must condition the number of accessible states and lead to the optimal state, if the temperature is lowered gradually in a slow and well controlled manner (as in the annealing technique) and towards a local minimum if the temperature is lowered abruptly [8]. The flowchart of the algorithm is shown in Fig. 2 [9].

If $\Delta E \leq 0$ (i.e. new state is better than the current state) the modification will be accepted, when $\Delta E > 0$ if the probability $exp(-\Delta E/T)$ is greater than a random number drawn uniformly from [0, 1], the modification, making the state worse, also will be accepted (Metropolis rule [8]).

By repeatedly observing the rule of acceptance, described above, a sequence of configurations (states) is generated, which constitutes a Markov chain (in a sense that each configuration depends on only that one which immediately precedes it) [8].

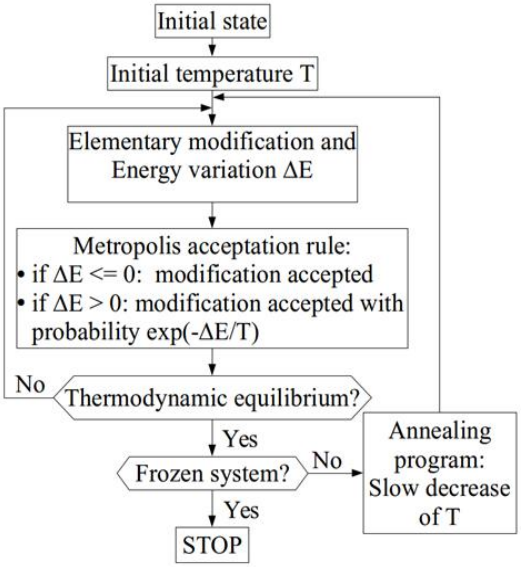

Fig. 2. Flowchart of SA [9].

IV. Parallel SA (PSA)

Parallel SA algorithm can be implemented in two forms: division and clustering.

A. Division Algorithm

Let *np* denote the number of processors (threads) in the parallel machine. A simple, effective way of parallelizing the SA algorithm consists in dividing the effort in generating the corresponding Markov chain (discussed later) over the *np* processors: each processor performing N_k/np trials. In order to keep the main characteristic of the SA algorithm, at each temperature level, when all processors finish processing their individual tasks, the incumbent optimal solutions are sent to the master node (*nodep = 0*), which then selects the best one and broadcasts the results to all other processors ($p = 1, \ldots, np - 1$).

The communication requirements of this algorithm are relatively small, which means that the potential for efficiency is high. The quality of the solution depends on the number of processors used in the parallel computation.

B. Clustering Algorithm

In this case, in contrast with the division algorithm discussed above, the sequential nature of the SA algorithm is strictly observed, as the *np* processors evaluate the N_k trials in a cooperative way, which means that all processors always work with the same current solution. Thus, whenever one processor accepts a new incumbent, it is communicated to all the other processors. This parallelization approach is less efficient at high-temperature levels where the frequency with which new current solutions are accepted is relatively high. The opposite happens at lower temperature levels where very few acceptances are performed and thus the algorithm presents best performance.

A good alternative consists in using a hybrid division/clustering algorithm. The process is started as a division algorithm and then switches to the clustering algorithm, which is executed for lower temperature levels.

The point at which the switching to the clustering algorithm occurs is based on the observed acceptance rate [10]. Here this hybrid manner is used.

V. The Hybrid Algorithm (PSO-PSA)

According to the diagram of our method (See Fig. 3), PSO and parallel SA algorithms were implemented consecutively. The objective functions of the two algorithms are the same. Diagram shows that PSO works with four particles. When PSO stopped (as a local maximum is being reached), the master process of SA takes state of the best particle from PSO, and after generating the initial temperature, it performs sufficient iterations (the number of Markov chains and their lengths), gives the slaves the temperature, current state, Markov length (divided by the number of processes for the division algorithm) and a variable that shows the parallelism type (division or clustering). In the following paragraphs, the important parameters and steps of the algorithm are represented.

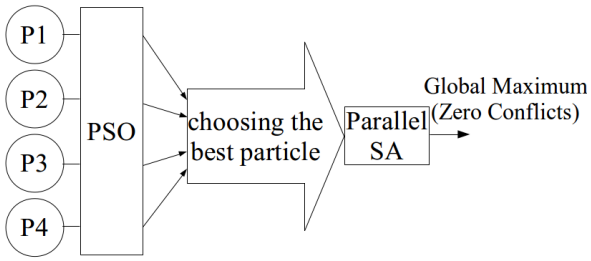

Fig. 3. The operation schema of the hybrid algorithm.

A. Representation of the Problem States

For adaptation between the problem and discrete PSO, problem states are represented by $\log(n) \times n$ binary matrices for $n \times n$ chess boards. Notice that this representation is used only when we want to update the state according to (4), whereas for other parts of the algorithm these state representations will be transformed to one dimensional matrices, by using a function with a time complexity of $O(n)$. Position of each queen in a chess board is shown in a binary matrix by representing the binary form of the rows' number in the corresponding columns. For example, the 4×4 chess board in Fig. 4a is represented as the 2×4 matrix shown in Fig. 4b, which will be used in updating time. For increasing the speed of calculations in evaluation procedure of the method the state shown in Fig. 4c will be used. In this transformed schema, each column holds the number of the row which the queens are located.

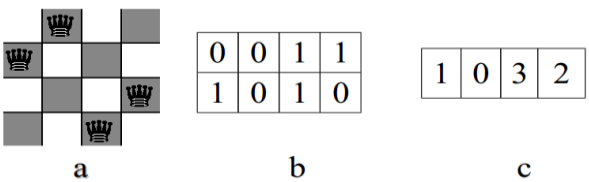

Fig. 4. Queens position representation.

B. Determining the Local Maximum of PSO

The algorithm traps in local maximum in PSO part when the evaluation of *nbest*, *pbest* and current state are equal for each particle for several numbers of iterations.

C. Number of Particles

Since there is no analytical method to determine the number of particles in PSO accurately, different numbers were tested, and we concluded that four particles could be appropriate for the problem, because in higher particle numbers, evaluation of *nbest*, *pbest* and current state don't decrease uniformly for each particle and in lower numbers the locality was reached too early.

D. Markov Chains

According to Fig. 2 the modifications of the states in each temperature stage should be sufficient to receive the thermal equilibrium (i.e. no more move can decrease the evaluation of the state) in that temperature. After the thermal equilibrium, the temperature decreases by a coefficient (e.g. 0.99). Here the number of temperature stages (Markov chains) is a function of dimension of the problem state, and the number of possible moves in each stage (Markov length) is the product of problem's dimension and the stage index.

The initial temperature, in the beginning of SA part, is generated by using the evaluation of the best state (*bestEvaluation*) resulted from PSO. The formula is as follows [10]:

$$T = (0.0001) \times (bestEvaluation) / -\log(0.15) \quad (5)$$

The coefficients are appropriate to avoid from an initial divergence of SA processes, i.e. the state doesn't go to very bad situations.

E. Steps of the Hybrid Algorithm

The algorithm consists of two parts (PSO and PSA). Detailed steps are as follows:

- Generate four particles with random initial states and velocities;
- Perform the updating procedures for each particle according to (4);
- If the local maximum is not being reached, go to 2; otherwise go to 4;
- Select the particle with the best state evaluation;
- Start master process of SA by using the best state;
- Generate the initial temperature $T0$ with (5);
- If the maximum number of the Markov chains is reached, go to 13, else go to 8 (see Fig. 5);
- Generate *np* parallel processes of SA (go to **Slave procedure**) and execute them;
- Wait until all processes are finished;
- If acceptance rate < threshold, then ptype = clustering (Here the threshold of acceptance rate is 0.0003);
- If ptype = division, state = the best state among all processes, else if ptype = clustering, state = the state of the process that is finished first;
- If evaluation is zero go to 13 else decrease the temperature (T) then go to 7;
- End of the algorithm.

Slave Procedure of SA:

1) Modify the state;
2) If the new state is accepted (according to the Metropolis

rule) and ptype = clustering, then stop all processes, return new state to the master process and go to 9, else go to 8. 3;

3) If the maximum Markov length is reached, then return the state to the master process and go to 9, else go to 8.1;

```
n = dimension; // number of Markov chains)
ptype = division; //parallelism type
//mcl is Markov chain length for each process;
for n = 0…n×n/2{
for m = 0 … np {
   if(ptype == division)
      mcl = (n+1) × dimension/np ;
   else   //Clustering Algorithm
      mcl = (n+1) × dimension ;
   process_m (mcl, T, ptype, state);
   …
   }
//Wait
   if (acceptance rate < threshold)
   ptype = clustering;
        Select the state and check the evaluation;
T = 0.99 × T;
}
```

Fig. 5. Pseudo code for PSA part of the algorithm.

VI. Experimental Results

Parallel section of PSO-PSA algorithm is implemented by multithreading method in java and tested by chess boards of different dimensions. The results are shown in Table I-Table IV and Fig. 6. All results are the average of 10 runs of the algorithms.

Table I shows the number of iterations required to achieve a good local maximum in the PSO part (i.e. when the evaluations of *nbest*, *pbest* and current state are equal for 25 times of updating).

Table II, Table III and Table IV represent the ratio of the number of iterations of PSO-PSA to that of PSA, PSO-SA and SA respectively, for solving six different dimensions of the problem, implemented by different number of threads. Finally, we can see a diagram representing the number of iterations for PSO-PSA tested by a different number of processes (threads).

From Table II, Table III and Table IV it is evident that PSO-PSA is more efficient than PSA, PSO-SA, SA, especially when the number of threads increases. This efficiency is also more significant in high dimensional problems, as it can be seen in the last three columns of Table II–Table IV where the number of iterations which PSO-PSA needs to solve the problem almost always is less than half the number of iterations which other methods need. The diagram shown in Fig. 6 represents that by increasing the number of processes, we gain more benefits of the parallelism approach especially in high dimensional problems.

TABLE I: Number of Iterations PSO Needs to Be Trapped in a Local Maximum

Dimension	20	100	200	500	800	1000	2000
PSO Iterations Number	353	1359	2489	5784	9602	11189	86592

TABLE II: Ratios of Number of Iterations of PSO-PSA to That of PSA for Different Dimensions

Number of Threads	Number of Queens (dimension)						
	20	100	200	500	800	1000	2000
3	1.32	0.70	1.38	0.80	0.35	0.43	0.53
4	1.33	1.06	0.93	0.78	0.47	0.45	0.33
8	1.16	0.93	1.08	0.77	0.44	0.47	0.52
15	0.77	1.16	1.08	1.17	0.79	0.45	0.27

TABLE III: Ratios of Number of Iterations of PSO-PSA to That of PSO-SA for Different Dimensions

Number of Threads	Number Of Queens (dimension)					
	20	100	200	500	800	1000
3	0.57	0.21	0.43	0.39	0.22	0.25
4	0.62	0.24	0.26	0.33	0.20	0.16
8	0.38	0.11	0.17	0.13	0.08	0.10
15	0.27	0.10	0.10	0.10	0.07	0.05

TABLE IV: Ratios of Number of Iterations of PSO-PSA to That of SA for Different Dimensions

Number of Threads	Number Of Queens (dimension)					
	20	100	200	500	800	1000
3	0.47	0.61	0.42	0.23	0.17	0.26
4	0.45	0.36	0.23	0.15	0.10	0.09
8	0.25	0.15	0.15	0.06	0.04	0.05
15	0.15	0.15	0.08	0.04	0.03	0.02

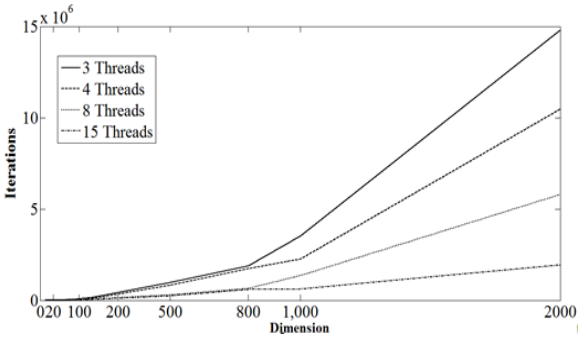

Fig. 6. Number of iterations of the proposed algorithm, implemented by different number of threads, for different dimensions of chessboard.

VII. Conclusion

The n-queens problem is good for testing new heuristic algorithms and comparing them with old ones. The results show that n-queens problem can be solved with a reasonable number of iterations using PSO-PSA in comparison with PSA, PSO-SA and SA by an increasingly ratio for higher dimensions. This means that when the dimension becomes larger PSO-PSA achieves to a global maximum (zero conflicts) faster.

The advantage of parallelism can be seen when thread numbers increases and this is important when the dimensions

of the problem increase.

The algorithm has many parameters such as the number of particles, enough PSO iterations, initial temperature, length of temperature stages, decreasing rate of temperature, number of processes, acceptance rate, etc., to be adjusted. Here, these parameters are determined by statistical tests. This work (adjusting parameters) can be done more precisely by other methods such as neural networks.

Although the proposed approach is introduced by testing it on solving n-queens problem, we are optimistic that this hybrid approach is useful for solving other CSP problems (e.g. Graph Coloring, Time Scheduling, etc.).

REFERENCES

[1] S. Russell and P. Norvig, *Artificial Inteligence: A Modern Approach*, 2nd ed., New Jersey: Pearson Education, 2003, ch. 5.

[2] X. Hu, R. C. Eberhart, and Y. Shi, "Swarm inteligence for permutation optimization: a case study of n-queens problem," in *Proc. the 2003 IEEE Swarm Intelligence Symposium*, 2003, pp. 243–246.

[3] M. Božikovic, M. Golub, and L. Budin, "Solving n-queens problem using global parallel genetic algorithm," *Eurocon 2003. Computer as a Tool, the IEEE Region 8*, vol. 2, pp. 104-407, 2003.

[4] J. Dréo, A. Pétrowski, P. Siarry, and E. Taillard, *Metaheuristics for Hard Optimization*, Berlin Heidelberg: Springer-Verlag, 2006, pp. 162-166.

[5] K. Y. Lee and M. Al-Sharkawi, *Modern Heuristic Optimization Techniques: Theory And Application To Power Systems*, Hoboken: Willey-Interscience, 2008, pp. 72-79.

[6] M. C., *Particle Swarm Optimization*, United States: ISTE, 2006, page: 39.

[7] K. Y. Lee and M. Al-Sharkawi, *Modern Heuristic Optimization Techniques: Theory And Application To Power Systems*, Hoboken: Willey-Interscience, 2008, p. 24.

[8] J. Dréo, A. Pétrowski, P. Siarry, and E. Taillard, *Metaheuristics for Hard Optimization*, Berlin Heidelberg: Springer-Verlag, 2006, pp. 25-31.

[9] J. Dréo, A. Pétrowski, P. Siarry, and E.Taillard, *Metaheuristics for Hard Optimization*, Berlin Heidelberg : Springer-Verlag, 2006, p. 8.

[10] K. Y. Lee and M. Al-Sharkawi, *Modern Heuristic Optimization Techniques: Theory And Application To Power Systems*, Hoboken: Willey-Interscience, 2008, ch. 1.

P. Jafarzadeh was born at Booshehr, Iran, on July 15, 1988 and received B.Sc. in computer engineering from Shahid Chamran University of Ahvaz, Ahvaz, Khouzestan, Iran in 2011.

He is now an independent scholar and attending for M.Sc. degree. He also works on Artificial Intelligence Programming. His research interests include metaheuristics, HPC and network security.

V. Mohammadi Saffarzadeh was born at Ahvaz, in Khouzestan, Iran, on April 28, 1988 and received B.Sc. in computer engineering from Shahid Chamran University of Ahvaz, Ahvaz, Khouzestan, Iran in 2011 and M.Sc. in artificial intelligence from Shahid Chamran University of Ahvaz, Ahvaz, Khouzestan, Iran in 2013.

He is now an independent scholar and also works on Artificial Intelligence Programming and Image Processing Projects. His research interests include metaheuristics, biomedical image processing, HPC and machine learning methods.