# Handling Infeasibility in Particle Swarm Optimization Using Constraint Programming

Robin Ajmera(✉), Akansha Kumar, and Sai K. Jayakumar

Jio Platforms Limited, Hyderabad, Telangana, India
robinajmera14@gmail.com

**Abstract.** Meta-heuristic approaches, particularly population-based methods, are used to solve NP-Hard problems in the search for near-optimal solutions. In highly combinatorial problems, due to the presence of numerous cons- traints, these population-based methods are expensive in terms of generating feasible solutions for the population across iterations. Several times, it is nearly impossible to get feasible solutions across pop ulations. Therefore, there is a need to reduce the infeasibility and refine the search space for these approaches. In general, meta-heuristic search capability needs to be increased when search space becomes discrete and is tight by constraints. In this paper, work has been done in that direction where search space is sampled from the feasible space prior to the application of Particle Swarm Optimization (PSO) such that it smartly moves in the feasible region and tries to find the optimal solution. The process starts with sampling using constraint programming with linear constraints and then applying a population-based meta-heuristic with a non-linear complex objective/fitness function including, penalizing infeasible solutions. The presented results are a testimony that our method is successful in reducing the infeasible region over the number of iterations.

**Keywords:** Optimization · Meta-heuristic · Particle swarm optimization · Genetic Algorithm · Constraint Programming (CP) · feasibility

## 1 Introduction

Optimization is a method to solve a problem to an optimal solution given some constraints and optimizing function. Optimization techniques are evolving and a lot of research is on the way to devise new algorithms and improve existing al- gorithms. Exact algorithms which always give the optimal solution have a short- coming in that they are expensive with respect to time complexity. Therefore, meta-heuristic techniques are devised which may or may not give the optimal solution but will at least give a good solution within a reasonable amount of time. There is a need for these techniques need to be improved by working in a direction to overcome their shortcomings. In this paper, work has been done to improve meta-heuristic population-based techniques by introducing feasible solutions using constraint programming to help algorithms search faster for solu- tions. To demonstrate our findings, the popular Vehicle Routing Problem (VRP) instance has been used with multiple vehicles having to travel to multiple nodes and return to the same origin node.


B. Raj et al. (Eds.): ICETE 2023, AER 223, pp. 378–388, 2023.
https://doi.org/10.2991/978-94-6463-252-1_42

## 2 Problem Description

Meta-heuristic approaches are used to solve NP-Hard Problems and the need is to solve them in a realistic and reasonable time. However, when the problem space is large with numerous constraints, the population-based approaches take a significant amount of time to even search for a feasible solution. To be specific, in population-based methods, while building (generating) the population (and future generations/iterations), due to the complexity of the search space, the samples are mostly infeasible (even up to 100%) in most of the problems. This leads to an astronomical increase in search time which affects the practical use of the solution. So, there is a need to get reduce infeasibility and refine the search space for population-based meta-heuristic approaches.

## 3 Literature Review

Research has been done to understand different population-based meta-heuristic methods, penalty functions, constraint satisfaction algorithms and different ap- proaches to constraint handling in heuristic methods.

Reference [1] Optimization problems with the help of genetic algorithms (GAs) and penalty functions were used for getting solutions in the feasible search space. Popular penalty functions with tuning parameters help to get better re- sults. Comparisons between feasible and infeasible solutions are made to find solutions in the feasible regions. Once sufficient feasible solutions are found, a good algorithm like GA with crossover and mutation approaches helps to get better solutions. Interestingly, diversity in the solutions was also maintained so that the searching capability is not restricted to a fixed zone of the feasible region.

Reference [2] Constraint handling is one of the major concerns when ap- plying genetic algorithms (GA) to solve optimization problems that are highly constrained. This paper proposes to use the gradient information derived from the set of constraints to correct infeasible solutions.

Reference [3] The Stochastic Ranking approach was proposed by Runars- son and Yao, and it uses a multi-membered evolution strategy. It was done by balancing objective functions and penalty functions. The approach requires a user-defined parameter called "probability function", which determines the bal-ance between the objective function and the penalty function.

Reference [4] Another paper presents a particle swarm optimizer for solving constrained optimization problems adopting a small population size. The selec- tion of particles is based on the distance of a solution from the feasible region.

Interestingly, a no-penalty function and initialization of the population were used in the framework.

Reference [5] This work researches in that direction where search space is sampled in the feasible region and after that population-based Particle Swarm Optimization (meta-heuristic techniques) is applied so that it smartly moves in the feasible region and tries to find the optimal solution. Reference [6] The first step involves sampling using constraint programming with linear constraints and then applying population-based meta-heuristic with a non-linear complex objective/fitness function including penalization of the infeasible solution.

In the draft contents, solution architecture and framework are proposed, fol- lowed by the description of the use case (scenario). Then follows the experiment setting and the results. We have set up our experiments for the Particle swarm optimization Framework and injected feasible solutions using Constraint Pro- gramming. Development of Generalized architecture for different population- based meta-heuristics is also proposed in our work. The paper ends with the conclusion along with potential possible future work.

## 4 The Particle Swarm Optimization (PSO) Algorithm

Reference [7], In the optimization problem, particularly in population-based metaheuristics like GA or PSO, a variable is defined by a vector,

$$X = [X_1, X_2, X_3, \ldots X_n] \tag{1}$$

that minimizes or maximizes a cost function f(X). It is an n-dimensional vector, where n represents the number of variables to be determined in a problem. For example, it can be considered as coordinates in the problem of being selected by a flock of birds based on their current position, individual movement, and the whole group's movement. The objective function's use is to get an idea about how good or bad a position X is. The following parameters are assumed:

f: objective function,

$X_i$: position of the particle or agent, $V_i$: velocity of the particle or agent, $A$: population of agents,

$W$ : inertia weight,

$C_1$: cognitive constant,

$C_2$: social constant,

$U_1$, $U_2$: random numbers,

$P_b$: personal best,

$g_b$: global best.

The actual algorithm works as follows:

1. Create a 'population' of agents (particles) that are uniformly distributed over X.
2. Evaluate each particle's position considering the objective function z.
3. If a particle's current position is improved from its previous best position, update it.
4. Find the best particle (which is at the best position based on the objective function).
5. Reference [8], Update particles' velocities.

$$V_i^{t+1} = W.V_i^t + c_1 U_1^t * (P_{b_i}^t - P_i^t) + c_2 U_2^t (g_b^t - P_i^t) \tag{2}$$

6. Particles get moved to a different location.

$$P_i^{t+1} = P_i^t + v_i^{t+1} \tag{3}$$

7. Go to step 2 until the stopping criteria are satisfied.

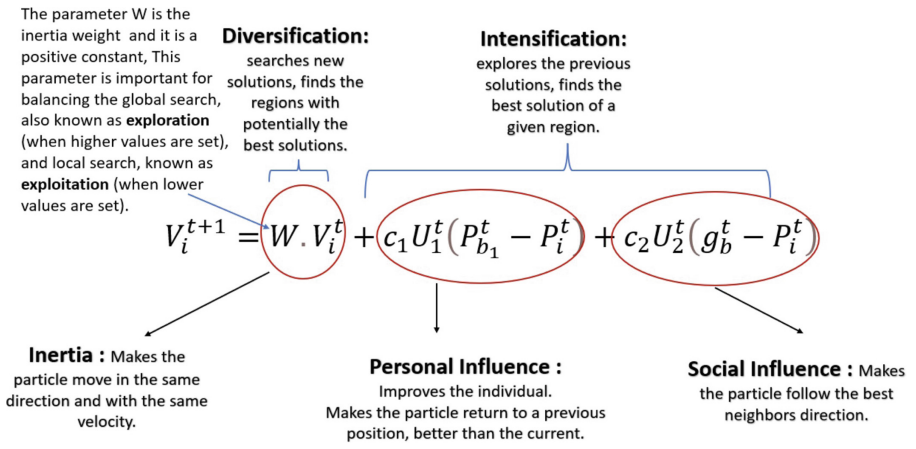


**Fig. 1.** Analysis of the Particle Swarm Optimization Algorithm

## 5 Analysis of PSO

Reference [7], As shown in Fig. 1, the significance of velocity updating equation parameters is as follows:

W: It is an inertial factor for velocity update. The particle's motion is de- pendent on the previous particle's motion, so the particle tends to move in the same direction.

C1: The parameter C1 existing as a product is a positive constant, and it is an individual-cognition parameter. It is the factor that is based on the particle's own previous experiences.

C2: It is a social learning parameter, and it takes care of the global learning of the swarm.

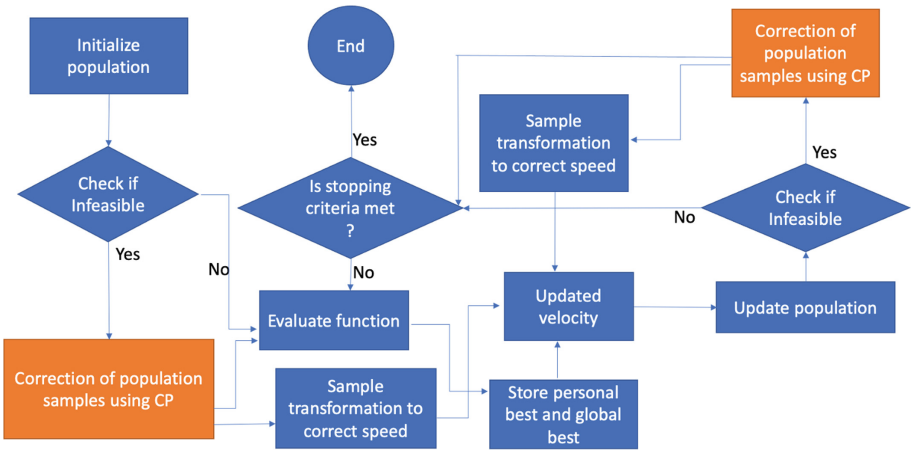


**Fig. 2.** Flowchart for PSO + CP

# 6 The Artwork

## 6.1 Architecture & Framework

Figure 2 presents a macroscopic view of the modified PSO algorithm which includes the CP correct step.

The following steps summarize the architecture:

1. Initialize the population of particles. This would be generated randomly. In scenarios, where a prior model provides a feasible solution, the same can be used as part of the initial population.
2. Check the feasibility of samples in the population using the constraints in- volved.
3. Certain parts of infeasible solutions are corrected using Constraint Program- ming (CP). This is the **novelty** we have presented in this work. In Fig. 2, this is shown as the orange boxes.
4. Evaluate the population using the evaluation (objective) function on each and every sample by penalizing the infeasible solution. The type of penalizing function has the potential for future work.
5. Store personal best and global best parameters based on the result of the evaluation.
6. Update the population using the PSO equation. In this step, we update the population based on the learnings from the previous iterations.
7. Again, check the feasibility of samples in the population using the constraints involved.
8. Again, certain parts of the infeasible solutions are corrected using Constraint Programming (CP). This is the novelty we have presented in this work. In Fig. 2, this is shown as the orange boxes.
9. In this step, the velocity is corrected from the corrected sample.
10. Repeat steps from the evaluation function (Step 4).
11. Stop evaluating based on the stopping criteria. In our case, when the time limit or the number of maximum generations, whichever is earlier.

## 6.2 Problem Scenario

The following scenario has been simulated to test our invention. M vehicles need to serve the demand in N cities by moving products from one depot. The coordinates (lat-long) of the depot and the cities are known.

1. Given N = 102 cities, M = 10 Vehicles and 1 Depot.
2. Latitude and longitude of different cities are known.
3. M Vehicles have to serve N cities such that a city is served once by the vehicles. The objective is to minimize the total distance traveled by the vehicles.

## 6.3 Formulation

1. Number of cities, N = 102.
2. Number of depots = 1.
3. Number of vehicles, M = 10.
4. Number of variables = N*M = 1020.

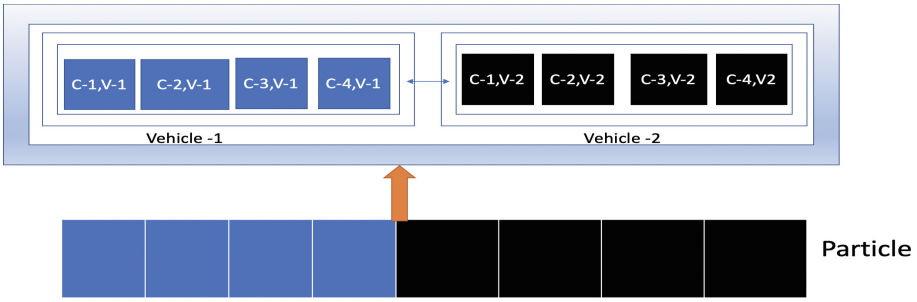


**Fig. 3.** Particle visualization

5. Number of constraints = N = 102.
6. Size of particle = Number of variables.
7. Particle Definition P: a particle is a vector of binary elements as denoted in Fig. 3. Each element denotes whether $m^{th}$ vehicle visits $n^{th}$ city or not.

$$\text{X[city]} = \begin{cases} 1 \text{ if city is visited by m}^{\text{th}} \text{ vehicle} \\ 0 \text{ if city is not visiited by m}^{\text{th}} \text{ vehicle} \end{cases} \tag{4}$$

8. Particle length $L$ is the number of cities times the number of vehicles which is the length of variables.

$$L = N * M \tag{5}$$

9. Constraint: A city is visited by one vehicle only

$$\sum_{m} P[city] = 1 \cdots \forall cities \tag{6}$$

10. The objective is to minimize the total distance $D$ traveled by all vehicles and the penalty which is the number of rejected samples in the population for any constraint violation $P$ in each iteration of the particle swarm optimization algorithm. Note $M$ is a large number to penalize.

$$Objective = Minimize\ (D + P * M) \tag{7}$$

## 7 Experiment and Results

Different cases have been experimented with using the DEAP library in Python by applying PSO and *CP SAT* solver for constraint programming in Python is used for solving the problem.

In the experimentation Table 1, we can observe that rejected samples are getting reduced significantly - more than 40% for this problem instance with different % of sample correction. This can be compared to the case without any sample corrections, where it leads to no reduction in infeasibility.

It is found that using extraction of feasible solutions at every iteration helps in getting a reduced number of infeasible solutions in the next iterations. In PSO without sample

**Table 1.** Experiment and Result.

| S.No. | Cases | % of feasible ini- tial population | %of feasible updated population | % Reduction In Infeasibility | Fitness |
|---|---|---|---|---|---|
| 1 | Without sample correction | NA | NA | NA | 1.00009 e + 12 |
| 2 | With 20% Sample correction using CP | 0 | 20 | 60 | 275386 45.899 |
| 3 | With 50% Sample correction using CP | 0 | 20 | 41.15 | 27758178. 09 |
| 4 | With 100% Sample correction using CP | 0 | 100 | 45.8 | 2709898 5.59 |

correction case, there is no reduction in infeasible solutions as also shown in Fig. 4. This case has no feasible solutions injected into the PSO. So, every time only infeasible solutions are generated and hence a horizontal straight line is obtained as shown in Fig. 4.

The same parameters are considered for PSO i.e. w (inertial parameter) = 0.7, c1 (cognitive parameter) = 0.4, c2 (social parameter) = 0.4, population size = 2000, as well as the time limit (600 sec) for running the algorithm for all cases. Percentage passing for sample correction is not applicable for VRP using PSO without sample correction as tabulated in Table 1.

In Figs. 5 and 7, it shows how the infeasibility is reducing over the itera- tions for 100% sample correction and 20% sample correction respectively. Also, in Figs. 6 and 8, there is an improvement in the fitness value over the number of iterations for 100% sample correction and 20% sample correction respectively.

## 8 Summary

The methods can be generalized to any population-based heuristics as shown in Fig. 9. Most of the flow will remain the same and the constraint correction activity can be replicated.

References [9] and [10], it is evident from the Fig. 9 that the customiza- tion happens in the updation block. In PSO (our case) it is the velocity, in Ant Colony Optimization (ACO) it is Pheromone updation and in Genetic Algo- rithms (GA) it is chromosome updation (crossover/mutation).

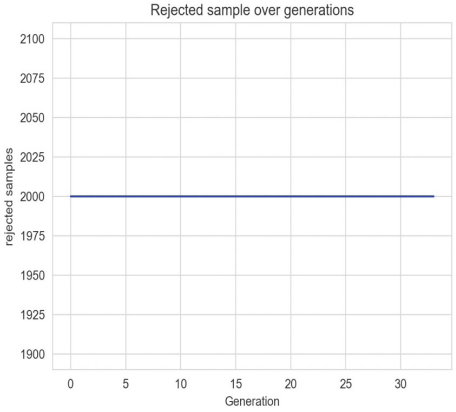


**Fig. 4.** PSO without sample correction: Sample rejected over iterations

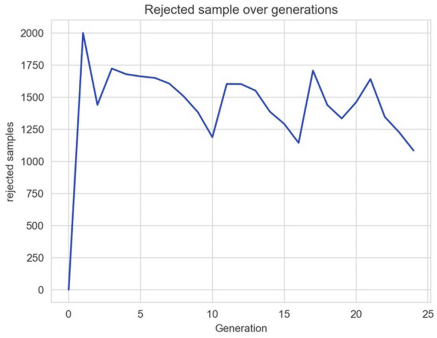


**Fig. 5.** PSO with 100% sample correction: Sample rejected over iterations

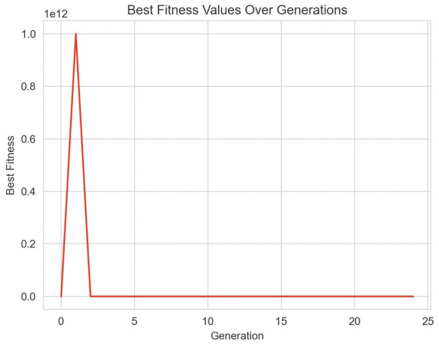


**Fig. 6.** PSO with 100% sample correction: Fitness value over iterations

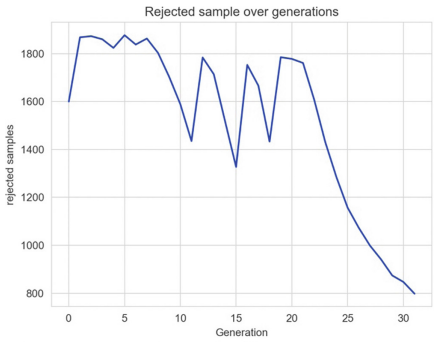


**Fig. 7.** PSO with 20% sample correction: Sample rejected over iterations

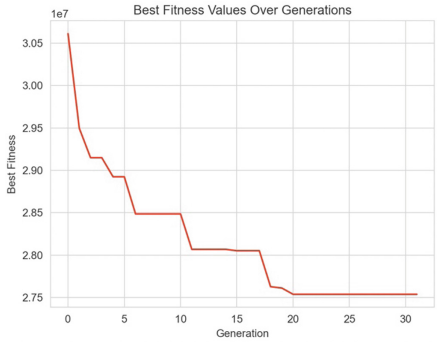


**Fig. 8.** PSO with 20% sample correction: Fitness value over iterations

## 9 Conclusion

In this work, we have shown a CP-based sample collection methodology to reduce the in-feasibility during the sample generation (population) process in PSO.

This Fig. 9. Flowchart For generalized population-based methods method, with minimal customization, is replicable across other population-based heuristic methods.

1. Reduction in the infeasible samples proves that a feasible set of solutions sampling using CP along with penalizing function drives the PSO algorithm to search for a solution in a feasible region.
2. Also, decrease in rejected samples over the generations/iterations shows that it helps PSO by its swarm intelligence to learn to find solutions in the feasible regions.
3. In practical applications, this method helps in the enhancement of search speed, leading to a computationally favorable solution for complex problems.

## 10 Future Work

The future work will involve the following two dimensions,

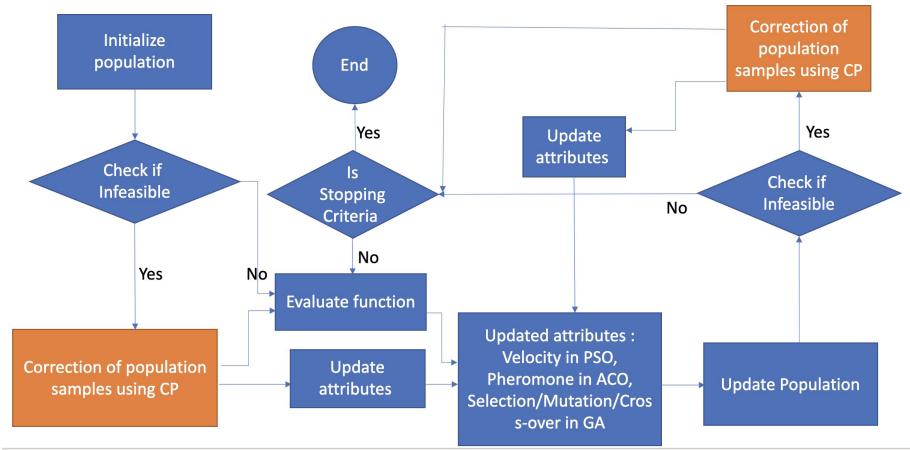


**Fig. 9.** PSO with 20% sample correction: Fitness value over iterations

4. Reference [11], Implement the CP-based method in other population-based meta-heuristic methods such as GA, ACO, Bee Colony Optimization (BCO) etc. Follow the same approach as presented in Fig. 9
5. Explore other penalty functions and the implementation to increase the speed of infeasibility correction, leading to faster/quicker search and better optimal solutions.
6. The current solution accounts for linear constraints only. Therefore, there is a need to work on getting feasible solutions for non-linear constraints as well. Implementation of the current methodology with sophisticated hard- ware may result in better solutions.

## References

1. Kalyanmoy Deb. An efficient constraint handling method for genetic algorithms. *Computer methods in applied mechanics and engineering*, 186(2–4):311–338, 2000.
2. Piya Chootinan and Anthony Chen. Constraint handling in genetic algorithms us- ing a gradient-based repair method. *Computers & operations research*, 33(8):2263– 2281, 2006.
3. Thomas P. Runarsson and Xin Yao. Stochastic ranking for constrained evolution- ary optimization. *IEEE Transactions on evolutionary computation*, 4(3):284–294, 2000.
4. Juan C Fuentes Cabrera and Carlos A Coello Coello. Handling constraints in par- ticle swarm optimization using a small population size. In *Mexican International Conference on Artificial Intelligence*, pages 41–51. Springer, 2007.
5. Sally C Brailsford, Chris N Potts, and Barbara M Smith. Constraint satisfaction problems: Algorithms and applications. *European journal of operational research*, 119(3):557–581, 1999.
6. Francesca Rossi, Peter Van Beek, and Toby Walsh. Constraint programming. *Foundations of Artificial Intelligence*, 3:181–211, 2008.
7. Subhasis Sanyal. An introduction to particle swarm optimization (pso) algorithm, Oct 2021.
8. Feng Chen, Xinxin Sun, Dali Wei, and Yongning Tang. Tradeoff strategy between exploration and exploitation for pso. In *2011 seventh international conference on natural computation*, volume 3, pages 1216–1222. IEEE, 2011.

9. Nada MA Al Salami. Ant colony optimization algorithm. *UbiCC Journal*, 4(3):823– 826, 2009.
10. Akansha Kumar and Pavel V. Tsvetkov. A new approach to nuclear reactor design optimization using genetic algorithms and regression analysis. *Annals of Nuclear Energy*, 85:27–35, 2015.
11. Du˘san Teodorovi´c. Bee colony optimization (bco). In *Innovations in swarm intel- ligence*, pages 39–60. Springer, 2009.