Probability Collectives: A multi-agent approach for solving combinatorial optimization problems

Anand J. Kulkarni, K. Tai*

School of Mechanical and Aerospace Engineering, Nanyang Technological University, 50 Nanyang Avenue, Singapore 639798, Singapore

1. Introduction

Traditionally, complex systems have often been seen as centralized systems, but as the complexity grew, it became necessary to handle the systems using a distributed and decentralized optimization approach. In a distributed and decentralized approach, the system is divided into smaller subsystems and optimized individually to get the system level optimum. Such subsystems together can be seen as a collective, which in other words is a group of learning agents. These agents are self-interested and work in some definite direction to optimize their local rewards or payoffs optimizing the global/system objective. The system objective is also referred to as the world utility and is a measure of performance of the whole system. Probability Collectives (PC) theory is a broad framework for modeling and controlling distributed systems, and it has deep connections to Game Theory, Statistical Physics, and Optimization [1].

The method of PC theory is an efficient way of sampling the joint probability space, converting the problem into the convex space of probability distribution. PC considers the variables in the systems as individual agents/players of a game being played iteratively [2]. Unlike stochastic approaches such as Genetic Algorithms (GA), Swarm Optimization and Simulated Annealing (SA), rather than deciding over the agent’s moves/set of actions, PC allocates probability values to each agent’s moves. In each iteration, every agent independently updates its own probability distribution in selecting a particular action out of its strategy set having the highest probability of optimizing its own utility (private utility) which also results in optimizing the world utility or system objective [1]. This is based on the prior knowledge of the actions/strategies of all the other agents.

In the approach presented in this paper, the system is divided into subsystems and each subsystem is considered as an agent. The framework of Collective Intelligence (COIN) is an approach to design a collective, which includes the computational agents working to enhance the system performance. Essentially, in the COIN approach, the agents select actions over a particular range and receive some rewards on the basis of the system objective achieved because of those actions. The process iterates and reaches equilibrium when no further increase in reward is possible for the
incorporating uncertainty. The PC approach was implemented by the authors of this paper in their earlier work [4] designing a stepped beam which produced encouraging results. Also, the PC approach was modified by the authors and successfully optimized the Rosenbrock function as a Multi-Agent System (MAS) [5]. This paper extends the work on the modified PC approach solving two test cases of the NP-hard combinatorial optimization problems like Multi-Depot Multiple Traveling Salesmen Problem (MDMTSP) with 3 depots, 3 vehicles and 15 nodes. The algorithm is accompanied with insertion, elimination and swapping heuristic techniques. Both the test cases produced optimum results in a reasonably short time.

This paper is organized as follows. Section 2 describes the various areas in which PC is implemented along with the related advantages. The formulation of its modified version is in Section 3. PC applied to optimize Rosenbrock function along with a discussion on the results is in Section 4. Section 5 briefly reviews the Multiple Traveling Salesmen Problem (MTSP), in which the important methods and techniques used to solve variations of the MTSP and the Vehicle Routing Problem (VRP) are discussed. Furthermore, the solution to Multi-Depot MTSP (MDMTSP) using PC and some heuristic techniques along with the results to two test cases are discussed in detail. It is followed by a useful discussion in Section 6, and concluding remarks and future work are in Section 7.

2. Review of PC

As PC is used on the distributed and decentralized platform, it is applied to variegated areas such as wireless sensor networks, scheduling, logistics and mechanical design.

The joint optimization of the routing and resource allocation in wireless networks was solved by some researchers [6–9]. In all these works, as the concave utility functions were difficult to solve using traditional optimization techniques, they were assumed to be monotonically increasing, strictly convex, positive and differentiable. On the other hand, the PC approach implemented in [10–12] considered arbitrary utility functions and played an important role by turning the original problem into convex optimization over a probability distribution.

In case of mobile wireless networks (ad hoc networks), choosing the optimal number of nodes for a cluster and choosing the cluster head is a NP-hard problem. The Greedy Election method was applied to this problem in [9] and resulted in suboptimal solution with unbalanced clustering of the nodes. The use of PC approach was demonstrated in [12] for the optimum selection of the cluster heads using the Weighted Clustering Algorithm. The nodes acted as quasi-autonomous agents participating into clusters independently. The results claimed were far better than the traditional benchmark approach of SA [13], and Greedy Election method [9] with regard to the distribution of nodes and balance of the cluster sizes. Moreover, as the Greedy Election method and SA are centralized approaches their performance may worsen with increase in the problem size.

In a complex system such as airplane fleet assignment with 129 variables and 184 constraints, the minimization of the number of flights was achieved using PC [2]. Applying a centralized approach to this problem may increase the communication and computational load. Further, it may add latency in the system resulting in the growing possibility of conflict in schedules and continuity. Using PC the airplanes adjusted their own schedules depending upon the individual payoffs for the possible routes, exploiting the advantages of distributed and decentralized approach [2]. The fleet assignment and aircraft routing problem was solved using centralized approaches with a very few number of constraints in [14,15] using integer programming and neighborhood search in [16] and using GA in [17].

The potential of PC in mechanical design was demonstrated for deciding the cross-sections of the individual bars and individual segments of a 10 bar truss [18] and a segmented beam [4], respectively. It is worth to mention here that the 10 bar truss problem in [18] was solved as a discrete constrained optimization problem, while the segmented beam problem in [4] was solved as an unconstrained continuous optimization problem. In both problems, possible values of the cross-sections of the individual components were the strategy sets. In [18], the solution was feasible but was worse than those obtained by other methods [19–21]. The probabilistic method such as GA was used in [19] where the constrained problem was converted into the unconstrained optimization problem using a penalty function approach. This method was modified by altering the parent selection and mutation probability to improve the results in [20]. In [21], a Region Reduction Simulated Annealing method was proposed. Simulated Annealing (SA) was used to search at multiple points iteratively through the population generated using GA. The search was done from within the most promising neighborhood region of possible candidate solution. This neighborhood concept is similar to the one used in the PC implementation in [4,5] and also discussed in this paper. Although in [21], it was claimed that the neighborhood concept helped to enhance the convergence speed; the additional computations were involved for cumulative distribution function and curve fitting, while on the other hand simple neighboring approach is used in the PC work presented here.

Benchmark problems such as Schaffer’s function, Rosenbrock function, Ackley Path function and Michalewicz Epistatic function were solved using GA and PC to test the important characteristics such as multimodality, nonlinearity and non-separability [22]. The results clearly indicated that GA was outperformed by PC in the rate of descent, trapping in false minima and long term optimization. The solutions to these test functions also indicated that PC can be applied to wide application areas. Moreover, at the core of the optimization algorithm such as GA is the population of solutions. In every iteration, each individual solution from the population is tested for its fitness to the problem at hand [22] and the population is updated accordingly. GA plots the best-so-far curve showing the fitness of the best individual in the last preset generations. On the other hand, in PC the probability distribution of the possible solutions is updated iteratively. PC plots the probability distribution of the available strategies across the variable space after a predefined number of iterations optimizing an associated maxent Lagrangian. It also directly incorporates uncertainty due to both imperfect sampling and the stochastic independence of the agents’ actions [22].

PC was evaluated in [23] using centralized and decentralized architectures solving the 8-Queens problem. The comparison between the two architectures clearly underlined the average superiority of the decentralized approach including the computational cost. This was because of the distributed sample generation and updating of the probabilities in the latter approach. In [23], PC was also compared with a backtracking algorithm referred to as Asynchronous Distributed OPTimization (ADOPT) [24]. The ADOPT algorithm being distributed, the communication and computational load was not equally distributed among the agents. It was also demonstrated that although ADOPT was guaranteed to find the solution in each run, communication and computations required were more than for the same problem solved using PC. It was demonstrated in [23] as well as in this paper that unlike ADOPT, due to probabilistic nature and randomness involved in the PC algorithm every run may not produce the same results even
though the initial conditions are identical. The highlighted advantages in [23] based on the results are listed in this paper as well.

As PC solves the problem as a MAS, it is worth to discuss some of the similarities and differences compared to the Multi-Agent Reinforcement Learning (MARL) methods. Most of the MARL methods such as fully cooperative, fully competitive and mixed (neither cooperative nor competitive) are based on Game Theory, Optimization and Evolutionary Computations [25]. According to [25], most of these types of methods possess less scalability and are sensitive to imperfect observations. Any uncertainty or incomplete information may lead to unexpected behavior of the agents. However, the scalability of the fully cooperative methods such as coordination free methods can be enhanced by explicitly using the communication and/or uncertainty techniques [26–28]. On the other hand, PC is scalable and can handle uncertainty in terms of probability. Moreover, the random strategies selected by any agent can be coordinated or negotiated with the other agents based on the social conventions, right to communication, etc. This social aspect makes PC a cooperative approach. Furthermore, the indirect coordination based methods work on the concept of biasing the selection towards the likelihood of the good strategies. This concept is similar to the one used in PC algorithm presented here, in which agents choose the strategy sets only in the neighborhood of the best strategy identified in the previous iteration. In case of the mixed MARL algorithms, the agents have no constraints imposed on their rewards. It is similar to the PC algorithm in which the agents respond or select the strategies and exhibit self-interested behavior. On the contrary to this similarity, the mixed MARL algorithms may encounter multiple Nash Equilibria while in PC a unique Nash Equilibrium can be achieved.

The above discussion shows that although PC is versatile and applicable to variegated areas, it has not yet been implemented to solve some combinatorial optimization problems such as the Multiple Traveling Salesman Problem (MTSP). The following section summarizes the advantages of PC when applied to the areas described in the literature discussed above.

2.1. Advantages of PC

The PC approach has the following advantages over the other tools that can be used in optimizing collectives:

1. PC is a distributed solution approach in which each agent independently updates its probability distribution at any time instance and can be applied to continuous, discrete or mixed variables, etc. [1,2,29]. The probability of the strategy set is always a vector of real numbers, regardless of the type of data under consideration. This exploits the techniques of the optimization for Euclidean vectors, such as gradient descent.

2. It is robust in the sense that the cost function (global/system objective) can be irregular or noisy, i.e. can accommodate noisy and poorly modeled problems [1,23].

3. The failed agent can just be considered as the one that does not update its probability distribution, without affecting the other agents. On the other hand, it may severely hamper the performance of the other techniques [23].

4. It provides the sensitivity information about the problem in the sense that a variable with a peaky distribution (having highest probability value) is more important in the solution than a variable with a broad distribution, i.e. peaky distribution provides the best choice of action that can optimize the global utility [1].

5. The minimum value of the global cost function can be found by considering the maxent Lagrangian equation for each agent (variable) [2].

6. The computational and communication load is marginally less and equally distributed among all the agents [23].

7. It can efficiently handle problems having large number of variables [2].

These advantages of PC make it a competitive choice over other algorithms. The key concept of PC theory is that the minimum value of the world utility/system objective can be found by considering the maximum entropy (maxent) of agent-variable.

3. PC framework

As mentioned previously, PC is a distributed, decentralized and cooperative algorithm implemented in the framework of COIN and has connections to Game Theory, Statistical Physics, and Optimization [2,18,29]. It was originally proposed by Dr. David H. Wolpert in 1999 in a Technical Report presented to NASA [29]. Thereafter, more work has been done to extend PC and its applications [4,5,9–12]. The PC framework is outlined in this section.

PC considers the variables in the system as individual self-interested learning agents/players of a game being played iteratively [2]. While working in some definite direction, these agents select actions over a particular range and receive some local rewards on the basis of the system objective achieved because of those actions. In other words, these agents optimize their local rewards or payoffs, which also optimize the system level performance. The process iterates and reaches equilibrium (referred to as Nash Equilibrium) when no further increase in the reward is possible for the individual agent by changing its actions further.

Moreover, the method of PC theory is an efficient way of sampling the joint probability space, converting the problem into the convex space of probability distribution. PC allocates probability values to each agent’s moves, and hence directly incorporates uncertainty. In each iteration, every agent independently updates its own probability distribution to select a particular action out of its strategy set having the highest probability of optimizing its own utility (private utility), which also results in optimizing the world utility or global system objective [3,1]. In other words, the agent finds the highest probability strategy value which contributes the most to the minimization of the system level objective. This is based on prior knowledge of the action/strategy sets of all other agents. In short, the agents in a PC framework need to have the knowledge of the environment along with the other agents’ probable strategies.

In some of the applications, the agents are also needed to provide the knowledge of the inter-agent-relationship. It is one of the information/strategy sets which every other entitled agent is supposed to know. There is also global information that every agent is supposed to know. This allows all the agents to know the right to access to the strategy sets. All of the decisions are taken autonomously by each agent considering the available information in order to optimize the local utilities and hence to achieve the optimum global goal or system objective. The following section discusses the PC procedure in detail.

3.1. Detailed PC algorithm

The detailed implementation of the modified PC theory is as follows, with the algorithm flowchart represented in Fig. 1.

Consider a problem comprising $N$ agents with each agent $i$ given a strategy set $X_i$, represented as

$$X_i = \{X_i^{[1]}, X_i^{[2]}, X_i^{[3]}, \ldots, X_i^{[m]}\}, \quad i \in \{1, 2, \ldots, N\}$$

(1)
where \( m_i \) is the number of strategies for agent \( i \). The detailed steps of the modified PC theory are as follows:

1. Assign uniform probabilities to the strategies of agent \( i \). This is because, at the beginning, the least information is available (largest uncertainty) about which strategy is favorable for minimization of the objective function at the particular iteration under consideration. Therefore, at the beginning of the ‘game’, each agent’s every strategy has probability \( 1/m_i \) of being most favorable. Therefore, probability of strategy \( r \) of Agent \( i \) is \( q(X_1^{ri}) = 1/m_i, \ r = 1, 2, \ldots, m_i \) (2)

Agent \( i \) selects its first strategy and samples randomly from other agents’ strategies as well. This forms a ‘combined strategy set’ represented as

\[ Y_i^{\bullet} = \{X_1^{ri}, X_2^{ri}, \ldots, X_{N-1}^{ri}, X_N^{ri}\}, \ r = 1, 2, \ldots, m_i \] (3)

Superscript \( \bullet \) indicates that it is a ‘random guess’ and not known in advance. It is important to note that every agent \( i \) forms \( m_i \) combined strategy sets and each one is denoted as \( Y_i^{ri} \).

From each of the combined strategy sets \( Y_i^{ri} \) for agent \( i \), compute the ‘expected local utility’ for each agent \( i \) with strategy \( r \) as [29]

\[ \text{Exp Utility of Agent } i \text{'s } r \text{'th strategy} = G(Y_i^{ri})q(X_i^{ri}) \prod_{(i)} q(X_{(i)}^{ri}) \] (4)

where \( (i) \) represents every agent other than \( i \), and \( G \) is world utility/system objective.

Compute the Expected Global Utility on the basis of these combined strategy sets for every agent \( i \). This is problem dependent. Update the probabilities of all the strategies of each agent \( i \) using Boltzmann’s temperature \( T \) as follows [29]:

\[ q(X_1^{ri}) = q(X_1^{ri}) - \alpha_{\text{step}} \cdot q(X_i^{ri})k_{\text{update}} \] (5)

where

\[ k_{\text{update}} = \frac{\text{(Contribution of Agent } i \text{'s } r \text{'th strategy})}{T} + S(q) + \ln(q(X_i^{ri})) \] (6)
and
\[
\text{Contribution of Agent } i^{[r]} = \text{Exp Utility of Agent } i^{[r]} - \text{Exp Global Utility of Agent } i
\]  
(7)

The step size \( \alpha_{\text{step}} \) is held constant throughout the optimization. The Boltzmann's temperature \( T \) is reduced after every fixed number of iterations by multiplying it with some fixed factor \( \alpha_T \). The values of \( \alpha_{\text{step}} \) and \( \alpha_T (0 < \alpha_{\text{step}} \leq 1, 0 < \alpha_T \leq 1) \) are chosen based on preliminary trials with the algorithm.

2. Using the above second order update rule, the strategy having maximum contribution towards the minimization of the objective is separated from the other strategies i.e. its probability is increased iteratively.

For any agent \( i \), if strategy \( r \) contributes more towards minimization of the objective than other strategies, its probability increases with some amount greater than the other strategies. This updating continues for predefined number of iterations to get clear probability distribution showing the highest probability for the optimal strategy value. Compute the ‘entropy’ of every agent’s probability distribution [29] as
\[
S_1(q) = -\sum_{i=1}^{m}[q(X_i^{[r]})\ln q(X_i^{[r]})]
\]  
(8)

3. Entropy is the indication of the information availability. It increases when the information becomes clearer and reaches to maximum when information available also reaches to the maximum. When the entropy reaches the maximum, the probability distribution available clearly distinguishes every strategy’s contribution towards the minimization of the Expected Global Utility. A detailed discussion on entropy is available in [1].

Repeat the procedure from Step 2 through Step 3 for predefined number of iterations \( k \).

4. For each agent, identify the strategy which contributes the maximum in minimizing the global utility and refer to it as the ‘favorable strategy’.

For further iterations, store the following results:

4.1 The ‘favorable strategy value’
4.2 The ‘sampling matrix’ from which the favorable strategy is achieved

Sample around the favorable strategy to select neighboring values within the range as follows:

\[
\text{New Strategy Limits} = \text{Favorable Strategy} \pm (\lambda_k \times \text{Favorable Strategy})
\]  
(9)

5. Call the ‘most favorable strategy’ out of all stored, along with its corresponding sampling matrix.

Repeat the procedure from Step 1 through Step 5 for predefined number of iterations \( n \).

6. Sample around the ‘most favorable result’ to select neighboring values in the reduced range as follows:

\[
\text{New Strategy Limits} = \text{Favorable Strategy} \pm (\lambda_n \times \text{Favorable Strategy})
\]  
(10)

7. Output the ‘recent strategy’ values and the corresponding objective function.

Repeat the procedure from Step 1 through Step 7 until the convergence criterion is satisfied or for predefined number of iterations \( s \).

The values of \( \lambda_k \) and \( \lambda_n \), referred to as neighboring factors can be selected as 0.5 and 0.1, respectively. It is worth to mention that the convergence criterion in the present PC approach is the number of predefined iterations and/or there is no change in the final goal value for a considerable number of iterations. On the other hand, the convergence criterion in the PC approach originally presented in [2,18,29] and also by the authors of this paper in their previous work presented in [4] was the number of iterations for which there was no change in the highest probability value of the most favorable strategy. Furthermore, sampling in further stages of the PC approach presented here is done using the range selection in the neighborhood of the most favorable value in the particular iteration, while regression and data aging approach was used in the original PC approach.

According to [25–28], in order to achieve a Nash Equilibrium, every agent in a MAS should have the properties such as rationality and convergence. Rationality refers to the property by which every agent selects (or converges to) the best possible strategy given the strategies of the other agents. The convergence property refers to the stability condition i.e. a policy using which every agent selects (or converges to) the best possible strategy when all the other agents use their policies from a predefined class (preferably same class). The Nash Equilibrium is naturally achieved when all the agents in a MAS are convergent and rational. Moreover, a Nash Equilibrium is guaranteed when all the agents use stationary policies, i.e. those policies which do not change over time. It is worth to mention here that all the agents in the MAS proposed using PC algorithm exhibit the above mentioned properties. It is elaborated in the detailed PC algorithm discussed in the previous few paragraphs.

In any game there may be a large but finite number of Nash Equilibria present as it depends on the number of strategies per agent as well as the number of agents. It is essential to choose the best possible combination of the individual strategies selected by each agent. It is quite hard to go through every possible combination of the individual agent strategies and choose the best out of it that can produce a best possible Nash Equilibrium and hence the system objective.

As discussed in the detailed PC algorithm, in each iteration every agent selects the best possible strategy referred to as the favorable strategy given the possible strategies of the other agents. This information about its favorable strategy is made known to all the other agents. In addition, the global knowledge such as system objective value is also available to each agent. This clearly helps all the agents take the best possible informed decision in every further iteration. This makes the entire system ignore a considerably large number of Nash Equilibria but select the best possible one in each iteration. Furthermore, from this current Nash Equilibrium point the algorithm progresses to the next Nash Equilibrium point with better system objective. As the algorithm progresses, those ignored Nash Equilibria as well as the best Nash Equilibria selected at previous iterations would be noticed as inferior solutions.

This process continues until there is no change in the final solution, i.e. no new Nash Equilibrium has been identified that proves the current Nash Equilibrium to be inferior. Hence, the system exhibits stage-wise convergence to a unique Nash Equilibrium which is accepted as the final solution.

4. Results for Rosenbrock function using PC

There are a number of benchmark test functions for contemporary optimization algorithms like GAs and Evolutionary Computation. The Rosenbrock function is an example of a nonlinear function having strongly coupled variables and is a real challenge for any optimization algorithm because of its slow convergence for most optimization methods [30,31]. The Rosenbrock function with \( N \) number of variables is given by
\[
f(X) = \sum_{i=1}^{N-1}[100(x_i^2 - x_{i+1})^2 + (1 - x_i)^2]
\]  
(11)
The procedure explained in Section 3 is followed for every agent from within the pre-specified range. The starting results and reach convergence faster, 42 strategies are considered assigned to be different (as shown in Table 1). To get the refined allowable range for each variable (agent) is intentionally updating the temperature and probabilities of the agent are to reach convergence. In all the trials, the step parameters for reaching the global minimum.

In the context of COIN, the function variables are seen as a collection of agents with each variable represented as an autonomous agent. These agents compete with one another to optimize their individual values and ultimately the entire function value (i.e., global utility). The Rosenbrock function with 5 variables \((N = 5)\) is solved here in which each variable/agent randomly selected the strategy within the range of values specified to each variable. Although the optimal value of each and every variable \(x_i\) is 1, the allowable range for each variable (agent) is intentionally assigned to be different (as shown in Table 1). To get the refined results and reach convergence faster, 42 strategies are considered for every agent from within the pre-specified range. The starting probabilities assigned for each strategy selected randomly is uniform i.e. 1/42. The procedure explained in Section 3 is followed to reach convergence. In all the trials, the step parameters for updating the temperature and probabilities of the agent are \(\alpha_T = 0.9\) and \(\alpha_{\text{step}} = 0.098\), respectively. On completion of every 10 iterations the temperature is decreased by multiplying \(\alpha_T\) with the current temperature \(T\). The values of neighboring factors such as \(\lambda_k\) and \(\lambda_m\) are selected as 0.5 and 0.1, respectively.

The problem is coded in MATLAB 7.4.0 (R2007A) on Windows platform using a Pentium 4, 3 GHz processor speed and 512 MB RAM. The results of 5 trials are shown in Table 1. The results for Trial 5 are plotted in Fig. 2. For Trial 5, the value of the function at iteration 113 is accepted as the final value as there is no change in the function value for a considerable number of iterations. The variations in the function value and the number of function evaluations among the trials are due to the probabilistic nature of PC as well as the randomness in the selection of the strategies.

A number of researchers have solved the Rosenbrock function using various algorithms. Table 2 summarizes the results obtained by these various algorithms: Chaos Genetic Algorithm (CGA) [30], Punctuated Anytime Learning (PAL) system [31], Modified Differential Evolution (Modified DE) proposed in [32] and Loosely Coupled GA (LCGA) implemented in [33] are some of the algorithms demonstrated on the Rosenbrock function.

Referring to Table 2, every variable was assigned the identical range of allowable values, which is different from Table 1 where each variable is assigned different allowable ranges. Further, even with a larger number of variables, the optimal function values are mostly better in Table 1 compared to Table 2. Thus it can be said that the approach presented here produced fairly good comparable results to those produced by previous researchers. The further sections focus on the introduction, a short literature review of the Multiple Traveling Salesmen Problem (MTSP) and its solution using PC.

5. Multiple Traveling Salesmen Problem (MTSP)

A generalization of the well-known Traveling Salesman Problem (TSP) is the Multiple Traveling Salesmen Problem (MTSP). As the name indicates, there is more than one traveling salesman in the MTSP. The many different variants of the MTSP seem more appropriate for real-life applications [34,35] than the TSP. The literature on TSP and MTSP suggests that as compared to the former, the latter received little attention from researchers.

The generalized representation of the MTSP is as follows. Consider a complete directed graph \(G = (N,E)\), where \(N = \{1, 2, \ldots, n\}\) is the set of nodes or vertices or cities to be visited, hereafter

![Fig. 2. Results for Trial 5.](image)
referred to as nodes and $E$ is the set of arcs connecting the nodes. The number of salesmen located at the depot is $m$. Generally, node 1 is considered as the depot which is the starting and end point of the salesmen's journey. All the salesmen start from the depot, visit the remaining nodes exactly once and return to the same depot. While following these routes the total cost of visiting all the nodes is minimized. Generally, cost metric is defined in terms of time and/or distance to travel between the nodes, etc. [36–39].

The applications of the MTSP include logistics and manufacturing scheduling [40,41], delivery and pick-up scheduling [42,43], satellite navigation systems [44], and problems like overnight security problem [45]. The Vehicle Routing Problem (VRP) is the generalization of the MTSP, in which the number of vehicles to be deployed is also minimized along with the distance, time, etc. In the VRP, the traveling vehicle's capacity and the demand associated with each node is also considered. Depending on the applications and requirements, there are some variations to the above general definitions of the VRP and the MTSP. For these variations the readers are encouraged to refer to [34,35].

5.1. Related work on MTSP/VRP

There are some bio-inspired optimization algorithms such as Genetic Algorithm (GA), Ant Colony Optimization (ACO), Artificial Neural Network (ANN), and Particle Swarm Optimization (PSO) that have been used to solve variants of the TSP/MTSP/VRP. These algorithms are used in conjunction with various local improvement/heuristic techniques to jump out of the local minima and also to reduce the computational load and time.

Some of such local improvement techniques used are node insertion, swapping of nodes, r-opt technique, etc. In the node insertion technique, the vehicle routes are formed by iteratively adding nodes into the routes. In the swapping technique, once the vehicle routes are formed, the nodes belonging to them are exchanged for the improvement. In the r-opt technique, some of the edges belonging to two vehicle routes are exchanged. The possible number of operations exchanging $r$ edges is $n^r$. In order to reduce the number of operations and computational load, the number of edges $r$ is limited to 2 or 3. The local search technique such as tabu search uses the combination of the exchange of edges in a depot or within depots. Similar techniques such as swapping and insertion are also used in the PC work presented in this paper.

Some of the fundamental ways of solving the MTSP are expansion of the problem by converting it into the standard TSP and the simplification of the problem by using the approach of cluster-first-route-second. The work in [46] is one of the earlier ones to expand the MTSP by converting it into the standard TSP by introducing $m – 1$ imaginary bases/depots. This makes the total number of cities to be $n + m – 1$. This increases the problem size to almost double [46–49]. It also extends the cost matrix to $n + m – 1 \times n + m – 1$ [50,51], making the problem computationally tedious as compared to the standard TSP with the same number of cities. This becomes worse when the number of cities is too large. Such transformed TSP is attempted using the ACO [49], the GA [41] and the PSO [52] showing the declining performance with the increase in number of nodes. The ANN is also applied with this approach in [53] resulting in increased computational complexities and could converge to only a valid but not an optimal solution.

The simplification approach is one which reduces the computational load using the decomposition approach referred to as cluster-first-route-second [47,54–57]. The formation of the clusters of the nodes is based on the distance between them, with problem specific constraints such as turning radius limitations in the case of Unmanned Aerial Vehicles (UAVs) [55]. The routes formed in the individual clusters are modified for further local improvement [54]. The approach of using one Ant Colony to form the clusters of nodes and the second Ant Colony optimizing the routes in the clusters is followed in [58] solving VRP with time windows. The savings principle is used in [56] biasing the formation of clusters of the closely located nodes. Such an approach was found to be useful when used with the computationally expensive tools such as GA, reducing the possible permutations forming the chromosomes.

The GA when used for the MTSP/VRP needs special attention. This is because the crossover operation may produce an invalid offspring which requires chromosome repair work to be carried out but at the expense of the characteristics of the parent solutions. The cycle crossover between the best and the randomly selected parent is proposed in [41] increasing the possibility of retaining the characteristics of the best parent. The repair work after crossover in GA and local search in PSO is totally avoided in [59,52], respectively, by incorporating the penalty coefficient into the objective function. The repair work based on the cost effectiveness is implemented in the repair presented in this paper.

When used for the MTSP/VRP [53,60], the PSO generally becomes weak at the local search and needs special hybridization techniques or increase in the number of particles up to a certain limit, but this increases the computational load [60]. In case of the ACO, the increase in the complexity in coding the ant trails and the associated local and the global updating makes the convergence quite slow and uncertain as compared to GA [49].

ANNs are not good at handling capacity constraints and time window constraints and are essentially limited to spatial applications [61]. The Self-Organizing Map (SOM) is used to cluster/categorize/group unorganized data. This makes the SOM useful for both the MTSP [48] and VRP. As SOMs are computationally very expensive, the clusters need to contain quite few numbers of nodes restricting the problem size. In terms of the computational time they outperform the elastic net approach [62] which is heavily interconnected and takes quite long to update all the routes.

The problems like MTSP/VRP can be handled more efficiently using PC because of its various advantages, reducing the complexities and increasing flexibility. The next section proposes the PC solution to two test cases of the MTSP with three depots referred to as Multi-Depot MTSP (MDMTSP).

5.2. Solution to the Multi-Depot MTSP (MDMTSP) Using PC

In the proposed Multi-Depot MTSP (MDMTSP), the traveling vehicles are considered as autonomous agents. These vehicles are assigned the routes, where every route is considered as a strategy. These routes are formed from randomly sampled nodes. This is in contrast to the Rosenbrock function problem in Section 4, where every agent randomly selected strategies within the range of values specified to each agent. The sampling of routes is an important step in applying PC to the MTSP. The sampling procedure is explained in the following section by an illustrated example with 3 vehicles and 15 nodes.

5.2.1. Sampling

In problems like the MTSP, there is a strong possibility that some nodes may get sampled by more than one vehicle and also some may not get selected. This is avoided in the first sampling phase forming the individual agent routes. As there are 3 vehicles and 15 nodes, a grid of $3 \times 7$ is formed making sure that every vehicle will be assigned at least one node. This is shown in Table 3. The first node randomly selects a cell from the grid, thus assigning itself to the corresponding vehicle. The cell once selected becomes unavailable for further selection by the remaining nodes. This process continues until all the nodes are assigned to the vehicles. The nodes assigned to the vehicles together form
individual vehicle routes. These routes are represented as the strategies randomly assigned to the vehicles. According to the grid shown in Tables 3 and 4 nodes are assigned to vehicle 1 forming a route/strategy represented as D1-3-5-15-1-D1. Similarly, route/strategy formed by vehicle 2 is D2-4-8-11-10-13-D2 and by vehicle 3 is D3-6-9-2-12-14-7-D3, where D1, D2 and D3 represent the depots corresponding to the vehicles.

This process continues for 45 times forming 45 possible combinations covering all the nodes using three vehicles. Out of these 45 combinations, the first 15 are assigned to vehicle 1 forming 15 combined strategy sets. Similarly, the next 15 combined strategy sets are assigned to vehicle 2 and the remaining 15 to vehicle 3. An example of a combined strategy set for a vehicle is represented as

$$\text{CombRoutes} = \{3, 5, 15, 1, 4, 8, 11, 10, 13, 6, 9, 2, 12, 14, 7\}$$  \hspace{1cm} (12)

According to the PC framework, in the combined strategy set shown in Eq. (12), the first vector subset route D1-3-5-15-1-D1 is assumed to be selected by the vehicle 1 while the remaining two vector subsets are guessed by it as the strategy sets selected by the remaining two vehicles. In order to reduce the computational load and reach the convergence faster, the number of combined strategies/routes per vehicle can be limited to a small number (15 in the current example). This makes it clear that there are 15 strategies/routes considered per vehicle and they are assigned uniform starting probability values (1/15 to each route in the current example) by the corresponding vehicle.

5.2.2. Formation of intermediate combined route set

Solving this problem according to the algorithmic procedure explained in Section 3, every vehicle converges to a probability distribution clearly distinguishing the most favorable route i.e. the one with the highest probability value. As discussed in Section 5.2.1, every vehicle has a strategy set with randomly sampled routes as its strategies. These strategies are represented on the X-axis of the probability distribution plots shown in Fig. 3. The routes are denoted in the form R##. For example, R11 represents route 1 of vehicle 1, while R15 represents route 15 of vehicle 1. Similarly, R21 represents route 1 of vehicle 2 and so on.

As shown by an example illustration in Fig. 3, vehicle 1 converges to R16, vehicle 2 converges to R29 and vehicle 3 converges to R314. These most favorable routes are joined to form the ‘intermediate combined route set’ represented as \{R16, R29, R314\}. This intermediate combined route set (also referred to as the ‘intermediate solution’) has a high probability of being infeasible. The following four reasons are responsible for the infeasibility:

1. The total number of nodes in the intermediate solution is less than the total number of available nodes.
2. A node has been selected by more than one vehicle causing repetition error.
3. Some nodes are left unselected because of reason 2.
4. The total number of nodes in the combined strategy/route set formed from the most favorable routes is more than the total number of available nodes.

The above reasons arise because of the nature of the PC algorithm. The most favorable routes selected by each vehicle are generally from different combined strategy/route sets belonging to the individual vehicles. Reason 1 can be avoided by repeating the sampling process until the valid intermediate solution is achieved. The next couple of sections discuss the implementation of the heuristic techniques dealing with reasons 2–4. Although, applied to the entire intermediate solution, for simplicity and better clarity the heuristic techniques are discussed only for two vehicles’ most favorable routes.

5.2.3. Node insertion and elimination heuristic

The insertion heuristic is carried out in order to deal with reasons 2 and 3. Generally in the insertion technique a node is iteratively added to the available tour and checked for feasibility and improvement. The insertion technique is also carried out in
using Branch and Bound technique implementing Best-First Search for Unmanned Aerial Vehicle (UAV) path planning. The insertion heuristic steps 1–4 followed in this paper are explained below:

1. The number of nodes of the combined strategy/route set formed from the most favorable strategy/route set is counted.
2. The missing nodes are identified.
3. The repeated nodes are replaced by the missing ones based on the total cost improvement. It is illustrated in Fig. 4 with an example of 2 vehicles and 11 nodes. Fig. 4(a) indicates that nodes 5 and 10 are visited by both vehicles and node 11 is left unselected. Accordingly there are four ways to insert node 11 into a route. These are inserting it in place of the repeated node 10 or 5 along the route belonging to either vehicle. Fig. 4(b) shows the lowest cost combined route/strategy set among these four different ways.
4. The steps from 1 to 3 are repeated until all the available nodes are covered and appear in the intermediate solution. If the total number of nodes to be covered is equal to the total number of available nodes in the intermediate solution, it indicates that the solution is feasible and therefore accept the current solution and skip to step 6; otherwise continue with step 5.
5. In some cases it is found that although all the available nodes are covered, the total number of nodes in the intermediate solution is more than the total number of available nodes. This indicates that some nodes were visited more than once. The elimination heuristic steps are as follows:
   5.1 The repeated nodes are identified.
   5.2 For the inter-vehicle repetition, the elimination of the repeated node from the different vehicle routes is accepted if the new solution leads to improvement. It is illustrated in Fig. 5(a).

   Continuing from Fig. 4(b), there are two possible remedies to inter-vehicle repetition. The first one is assigning node 5 to the route belonging to vehicle 1 and second is assigning it to the route belonging to vehicle 2. Fig. 5(a) shows the better of the two options.
5.3 The intra-vehicle repetition is treated based on the elimination of the repeated node and the associated improvement in the feasible solution. Continuing from Fig. 5(a), there are two possible remedies to intra-vehicle repetition to eliminate the repeated node 6. The first one is forming the route of vehicle 1 as D1-5-7-8-6-9-10-D1 and the second one is D1-5-6-7-8-9-10-D1. The second option shown in Fig. 5(b) is the better one and valid as well. If the total number of nodes to be covered is equal to the total number of nodes in the intermediate solution, it indicates that the solution is feasible. This solution is accepted as the current solution. This way the most favorable routes are modified repairing the infeasible intermediate solution to convert it into the feasible intermediate solution.

5.2.4. Neighboring approach
The standard procedure of sampling in the neighborhood of the most favorable strategies is explained in Section 3. It is applied in the MDMTSP solution procedure only when the feasible solution is achieved. The feasibility here refers to the solution having all the nodes selected with no repetition and at least one node is selected by each vehicle. The corresponding heuristic techniques are discussed in Section 5.2.3.

Fig. 4. Insertion heuristic.

Fig. 5. Elimination heuristic.
In the solution of the Rosenbrock function discussed in Section 4, the values of the neighboring factors $\lambda_k$ and $\lambda_n$ were 0.5 and 0.1, respectively. Instead of using such factors, neighboring radii are used in the solution procedure of MDMTSP. For the first few iterations, sampling is carried out in the neighborhood of the node by considering very large neighboring radii covering all the nodes. This facilitates every vehicle to select any available node and helps the algorithm to have an expanded search. Unlike the standard procedure in Section 3 and also applied to the Rosenbrock function where the neighboring factors were changed on completion of the predefined number of $k$ iterations as well as $n$ iterations; here the neighboring radii are reduced progressively only when there is no change in the solution for a predefined number of iterations. It is important to mention here that an identical neighboring radius is applied to every node of a route corresponding to the particular vehicle. Moreover, the nodes covered in that neighboring zone are eligible to be selected by the vehicle for further iterations. An example with arbitrary neighboring radius at some arbitrary iteration is shown in Fig. 6. The circles around the nodes belonging to vehicle 1 represent the neighboring zones. The nodes (5, 6, 7, 8, 9, 10 and 11) in these zones are available to be sampled randomly for vehicle 1 for the next iteration.

The procedure from Section 5.2.1 to Section 5.2.4 is repeated until there is no change in the final solution or for a predefined number of iterations i.e. until convergence. On convergence, the node swapping heuristic technique is applied. This is discussed in the next section.

5.2.5. Node swapping heuristic

The heuristic of swapping the set of arcs is one of the techniques used in the multi-depot case. This technique is used in [63] and represented as inter-depot mutation exchanging a set of arcs. In the current work presented here, an approach similar to [63] but swapping the entire set of nodes is implemented when there is no change in the current solution for a predefined number of iterations. The entire sets of nodes belonging to two depots are exchanged and the improved solution is accepted. This heuristic proved to be very useful for the PC algorithm solving the two cases presented in the following sections. This heuristic also helped in jumping out of local optima. Continuing from the feasible solution represented in Fig. 5(b), the node swapping is demonstrated in Fig. 7.

5.3. Test cases of the Multi-Depot MTSP (MDMTSP)

PC accompanied with the above described heuristic techniques is applied to two test cases of the MDMTSP. Similar to the Rosenbrock function, the test cases are coded in MATLAB 7.4.0 (R2007A) on Windows platform using Pentium 4, 3 GHz processor speed and 512 MB RAM. Case 1 is presented in Fig. 8(a). Three depots are placed 120° apart from one another on the periphery of an inner circle. Fifteen equidistant nodes are placed on the outer
circle. The angle between the nodes is 24°. The diameters of the inner circle and the outer circle are 10 and 40 units, respectively. One traveling vehicle is assigned per depot. The vehicles start their journeys from their assigned depots and return to their corresponding depots. According to the geometry of the problem and also assuming that the problem is symmetric (i.e., the cost of traveling between any two nodes is the same in both directions), the true optimum traveling cost is attained when vehicle 1 at depot 1 traveled the route D1-3-4-5-6-7-D1, vehicle 2 at depot 2 traveled the route D2-8-9-10-11-12-D2, and vehicle 3 at depot 3 traveled the route D3-13-14-15-1-D3. The true optimum solution was achieved by the PC approach and is plotted in Fig. 9(a).

According to the coordinates of the nodes and the depots, the optimum cost of traveling each of these routes is 134.7870 units. Hence, the optimum total traveling cost is 404.3610 units. A total of 50 runs were conducted for Case 1 and the true optimum was reached in every run. The time to reach the optimum varied between 0.35 min and 3.83 min with an average time of 2.09 min.

Case 2 is presented in Fig. 9(a) where three depots are placed apart from one another on the periphery of an inner circle. Fifteen nodes are placed on the outer circle. These nodes are arranged in three clusters. In every cluster, the nodes are intentionally placed at uneven angles apart. The diameters of the inner circle and the outer circle are 5 and 10 units, respectively. Similar to Case 1, one vehicle is assigned per depot. The vehicles start their journeys from their assigned depots and return to their corresponding depots. According to the geometry of the problem and also assuming that the problem is symmetric, the optimum traveling cost is attained when vehicle 1 at depot 1 traveled the route D1-6-7-8-9-10-D1, vehicle 2 at depot 2 traveled the route D2-11-12-13-14-15-D2, and vehicle 3 at depot 3 traveled the route D3-1-2-3-4-5-D3. The true optimum solution was achieved by the PC approach and is plotted in Fig. 9(b).

According to the coordinates of the nodes and the depots, the optimum cost of traveling each of these routes is 20.4264 units. Hence, the optimum total traveling cost is 61.2792 units. For Case 1, the optimum cost of traveling each of these routes is 20.4264 units. The time to reach the optimum varied between 0.76 min and 3.10 min with an average time of 1.27 min.

6. Comparison and discussion

The above solution to the MDMTSP using PC indicates that it can successfully be used to solve the Multi-Agent Combinatorial Optimization problems. As shown in Table 4, the results are compared with those of some other methods solving the Single Depot MTSP with problem sizes close to the specially developed test cases solved here. It is worth mentioning that the technique of converting the MTSP into the standard TSP using Miliotis reverse algorithm [64], cutting plane algorithm [51], elastic net approach [48] as well as Branch and Bound techniques such as ‘branch on an arc’ and ‘branch on a route’ with different lower bounds (LB0, LB2, etc.) in [50] did not produce the optimum solution in every run. This requires running the algorithms for a number of trials and then accepting the solution that is the average from among the trials. On the other hand, every run of the PC converged to the actual optimal solution in quite reasonable CPU time. Furthermore, the CPU time varied within a very small range. The time variation exists because of the probabilistic nature of PC algorithm and random initial sampling of the routes affecting every subsequent step.

As per the short literature review in Section 5.1, the popular optimization techniques are computationally time consuming and expensive. It is also worth to mention that the above insertion, elimination and swapping heuristics could be easily accommodated into PC.

Furthermore, the PC approach presented in this paper is a modified version. The Monte Carlo sampling used in the original PC approach [2], is computationally expensive and slow as the number of samples may be in the thousands or even millions. On the contrary, for the modified PC approach presented here, the pseudorandom scalar values drawn from uniform distribution are fewer in number. In addition, the regression (necessary in the original PC approach to fit the individual utility inside the individual range domain) is completely avoided here, making the modified approach computationally less expensive. Most significantly, the sampling range of individual variables (agents) is narrowed down with every iteration, ensuring faster convergence and an improvement in efficiency over the original PC approach [5]. For the MDMTSP, the neighboring radius is narrowed down iteratively and thus exploits the same benefits.

Besides the advantages of PC, some weaknesses are also identified. The present PC approach cannot handle constraints. Some efforts to circumvent this have been taken in the current work, in which the basic constraints of the MTSP are treated using repair work, i.e., using various heuristic techniques. Such repair techniques may not be an elegant option since they cannot be used for handling generic constraints. If complexity of the problem and related constraints increase, the repair work may become more tedious and may add further computational load. In addition, the
number of function evaluations can become very large because of its dependence on the number of strategies in every agent's strategy sets. Both of these disadvantages may limit its use to smaller size problems with fewer constraints.

7. Concluding remarks and future work

The modified PC methodology is demonstrated successfully in the COIN framework. The authors believe that the algorithm is made simpler and faster by improving the sampling method, the convergence criterion and most importantly the neighboring approach narrowing the sampling options. It is also evident that the growing complexity can be easily handled by decomposing the system into smaller subsystems or agents. The algorithm worked well to yield the optimum solution to the Rosenbrock function. It is also evident that the heuristic techniques such as insertion, elimination and swapping can be easily accommodated into PC. With the help of these heuristic techniques, PC produced optimum results solving a combinatorial optimization problem such as the Multi-Depot Multiple Traveling Salesmen Problem (MDMTSP). The solutions are produced for two test cases of the MDMTSP in reasonably short time.

Future work will include the development of general constraint handling techniques that can be incorporated into the PC approach. Furthermore, in order to increase the efficiency of the algorithm, efforts will be taken to trim the number of function evaluations by reducing the dependence on the number of individual agent strategies. In addition, more realistic path planning problems of the Multiple Unmanned Vehicles (MUVs) can then be solved with the MDMTSP and VRP approaches. The authors also see the potential in fields such as machine shop scheduling with every agent having different local utility function.

Acknowledgements

The authors would like to acknowledge Dr. Rodney Teo and Ye Chuan Yeo (DSO National Laboratories, Singapore) for their useful discussions on this topic, as well as Dr. David H. Wolpert (NASA Ames Research Center, Moffett Field, CA, USA) and Dr. Stefan R. Bieniawski (Boeing Phantom Works, Seal Beach, CA, USA) for their help with the study of some of the Probability Collectives concepts.

References


