2

Differential Evolution for Permutation−Based Combinatorial Problems

Godfrey Onwubolu[1] and Donald Davendra[2]

[1] Knowledge Management & Mining, Inc., Richmond Hill, Ontario, Canada
onwubolu_g@dsgm.ca
[2] Tomas Bata Univerzity in Zlin, Faculty of Applied Informatics, Nad Stranemi 4511, Zlin 76001, Czech Republic
davendra@fai.utb.cz

Abstract. The chapter clarifies the differences between wide-sense combinatorial optimization and strict-sense combinatorial optimization and then presents a number of combinatorial problems encountered in practice. Then overviews of the different permutative-based combinatorial approaches presented in the book are given. The chapter also includes an anatomy of the different permutative-based combinatorial approaches in the book, previously carried out elsewhere to show their strengths and weaknesses.

2.1 Introduction

It is first necessary to define what combinatorial problems are. In combinatorial problems, parameters can assume only a finite number of discrete states, so the number of possible vectors is also finite. Several classic algorithmic problems of a purely combinatorial nature include sorting and permutation generation, both of which were among the first non−numerical problems arising on electronic computers. A permutation describes an arrangement, or ordering, of parameters that define a problem. Many algorithmic problems tend to seek the best way to order a set of objects. Any algorithm for solving such problems exactly must construct a series of permutations. [10] classify wide-sense and strict−sense combinatorial optimization.

2.1.1 Wide-Sense Combinatorial Optimization

Consider switching networks which can be divided into fully connected non-blocking networks, and fully connected but blocking networks. Non−blocking switching networks can be re−arrangeable non-blocking networks, wide-sense non-blocking networks, and strictly non-blocking networks. A network is classified as rearrangeable if any idle input may be connected to any idle output provided that existing connections are allowed to be rearranged. A strictly non−blocking network on the other hand is always able to connect any idle input to any idle output without interfering with the existing connections. Wide−sense non-blocking network achieves strictly non-blocking property with the help of an algorithm.

G.C. Onwubolu and D. Davendra (Eds.): Differential Evolution, SCI 175, pp. 13–34.
springerlink.com © Springer-Verlag Berlin Heidelberg 2009

Let us consider another example. In this case we consider nuts having pitch diameters which are expressed in decimal places such as 2.5 mm, 3.6 mm,..., 5.3 mm, 6.2 mm, etc. These nuts are grouped into classes so that Class A nuts belong to those nuts whose pitch diameters lie between 2.5 mm−3.6 mm, Class B nuts belong to those nuts whose pitch diameters lie between 3.65 mm−4.8 mm, etc. In this case the classes are wide sense but the actual dimensions are continuous in nature. Therefore picking nuts based on their classes could be viewed as a wide−sense combinatorial problem because dimensional properties of a nut are continuous variables.

2.1.2 Strict-Sense Combinatorial Optimization

There are a number of **strict-sense** combinatorial problems, such as the traveling salesman problem, the knapsack problem, the shortest-path problem, facility layout problem, vehicle routing problem, etc. These are **strict-sense** combinatorial problems because they have no continuous counterpart [10]. These **strict-sense** combinatorial problems require some permutation of some sort. The way the objects are arranged may affect the overall performance of the system being considered. Arranging the objects incorrectly may affect the overall performance of the system. If there is a very large number of the object, then the number of ways of arranging the objects introduces another dimension of problem known as combinatorial explosion. Classical DE was not designed to solve this type of problem because these problems have hard constraints. Strong constraints like those imposed in the traveling salesman problem or facility layout problem make **strict-sense** combinatorial problems notoriously difficult for any optimization algorithm. It is this class of problems that this book is aimed at solving. A number of techniques have been devised to stretch the capabilities of DE to solve this type of hard constraint-type problems.

2.1.3 Feasible Solutions versus "Repairing" Infeasible Solutions for Strict-Sense Combinatorial Optimization

In DE′s case, the high proportion of infeasible vectors caused by constraints prevents the population from thoroughly exploring the objective function surface. [10] concluded that in order to minimize the problems posed by infeasible vectors, algorithms can either generate only feasible solutions, or "repair" infeasible ones.

The opinion expressed in this book is that all good heuristics are able to transform a combinatorial problem into a space which is amenable for search, and that there is no such thing as an "all-cure" algorithm for combinatorial problems. For example, particle swarm optimization (PSO) works fairly well for combinatorial problems, but only in combination with a good tailored heuristic (see for example, [8]). If such a heuristic is used, then PSO can locate promising regions. The same logic applies to a number of optimization approaches.

2.2 Combinatorial Problems

A wide range of strict-sense combinatorial problems exist for which the classical DE approach cannot solve because these problems are notoriously difficult for any

optimization algorithm. In this section, some of these problems are explained and their objective functions are formulated. The knapsack problem, travelling salesman problem (TSP), drilling location and hit sequencing, dynamic pick and place (DPP) model in robotic environment, vehicle routing problem (VRP), and facility layout problem, which are examples of strict-sense combinatorial problems, are discussed in this sub-section.

2.2.1 Knapsack Problem

For example, the single constraint (bounded) knapsack problem reflects the dilemma faced by a hiker who wants to pack as many valuable items in his or her knapsack as possible without exceeding the maximum weight he or she can carry. In the knapsack problem, each item has a weight, w_j, and a value, c_j (Equation 2.1); the constraints are in (Equation 2.1). The goal is to maximize the value of items packed without exceeding the maximum weight, b. The term represents the number of items with weight w_j and value, c_j :

maximize:

$$\sum_{j=0}^{D-1} c_j x_j, \quad x_j \geq 0, \text{integers} \tag{2.1}$$

subject to:

$$\sum_{j=0}^{D-1} w_j x_j \leq b, \quad w_j \geq 0, \ b > 0. \tag{2.2}$$

The solution to this problem will be a set of integers that indicate *how many* items of each type should be packed. As such, the knapsack problem is a strict-sense combinatorial problem because its parameters are discrete, solutions are constrained and it has no continuous counterpart (only a whole number of items can be placed in the knapsack).

2.2.2 Travelling Salesman Problem (TSP)

In the TSP, a salesman must visit each city in his designated area and then return home. In our case, the worker (tool) must perform each job and then return to the starting condition. The problem can be visualised on a graph. Each city (job) becomes a node. Arc lengths correspond to the distance between the attached cities (job changeover times). The salesman wants to find the shortest tour of the graph. A tour is a complete cycle. Starting at a home city, each city must be visited exactly one time before returning home. Each leg of the tour travels on an arc between two cities. The length of the tour is the sum of the lengths of the arcs selected. Fig 2.1 illustrates a five-city TSP. Trip lengths are shown on the arcs in Fig 2.1, the distance from city i to j is denoted by c_{ij}. We have assumed in the figure that all paths (arcs) are bi-directional. If arc lengths differ depending on the direction of the arc the TSP-formulation is said to be asymmetric, otherwise it is symmetric. A possible tour is shown in Fig 2.2. The cost of this tour is c12 + c24 + c43 + c35 + c51.

Several mathematical formulations exist for the TSP. One approach is to let x_{ij} be 1 if city j is visited immediately after i, and be 0 if otherwise. A formal statement of TSP is given as follows:

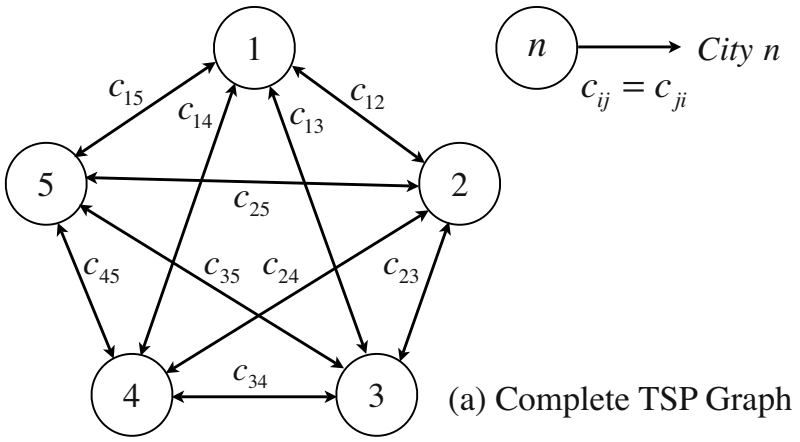

Fig. 2.1. (a) TSP illustrated on a graph

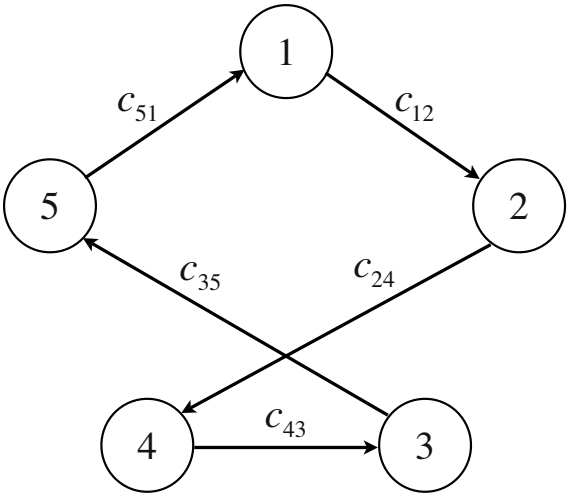

Fig. 2.2. (b) TSP illustrated on a graph

minimise

$$\sum_{i=1}^{N}\sum_{j=1}^{N} c_{ij}x_{ij} \tag{2.3}$$

subject to

$$\sum_{j=1}^{N} x_{ij} = 1; \forall i \tag{2.4}$$

$$\sum_{i=1}^{N} x_{ij} = 1; \forall i \tag{2.5}$$

No subtours

$$x_{ij} = 0 \text{ or } 1 \tag{2.6}$$

No *subtours* mean that there is no need to return to a city prior to visiting all other cities. The objective function accumulates time as we go from city i to j. Constraint 2.4 ensures that we leave each city. Constraint 2.5 ensures that we visit (enter) each city. A subtour occurs when we return to a city prior to visiting all other cities. Restriction 6.6 enables the TSP-formulation, differ from a linear assignment programming (LAP) formulation. Unfortunately, the non-subtour constraint significantly complicates model solution. One reasonable construction procedure for solving TSP is the closest insertion algorithm. This is now discussed.

The Traveling Salesman Problem (TSP) is a fairly universal, strict-sense combinatorial problem into which many other ***strict-sense*** combinatorial problems can be transformed. Consequently, many findings about DE′s performance on the TSP can be extrapolated to other **strict-sense** combinatorial problems.

2.2.2.1 TSP Using Closest Insertion Algorithm

The closest insertion algorithm starts by selecting any city. We then proceed through $N-1$ stages, adding a new city to the sequence at each stage. Thus a partial sequence is always maintained, and the sequence grows by one city each stage. At each stage we select the city from those currently unassigned that is closest to any city in the partial sequence. We add the city to the location that causes the smallest increase in the tour length. The closest insertion algorithm can be shown to produce a solution with a cost no worse than twice the optimum when the cost matrix is symmetric and satisfies the triangle inequality. In fact, the closest insertion algorithm may be a useful seed-solution for combinatorial search methods when large problems are solved. Symmetric implies $c_{ij} = c_{ji}$ where c_{ij} is the cost to go from city i directly to city j. Unfortunately, symmetry need not exist in our changeover problem. Normally, the triangular inequality ($c_{ij} \leq c_{ik} + c_{kj}$) will be satisfied, but this alone does not suffice to ensure the construction of a good solution. We may also try repeated application of the algorithm choosing a different starting city each time and then choose the best sequence found. Of course, this increases our workload by a factor of N. Alternatively, a different starting city may be chosen randomly for a specific number of times, less than the total number of cities. This option is preferred for large problem instances.

We now state the algorithm formally. Let S_a be the set of available (unassigned) cities at any stage. S_p will be the partial sequence in existence at any stage and is denoted $S_p = \{s_1, s_2, ..., s_n\}$, implying that city s_2 immediately follows s_1. For each unassigned city j, we use $r(j)$ to keep track of the city in the partial sequence that is closest to j. We store $r(j)$ only to avoid repeating calculations at each stage. Last, bracketed subscripts $[i]$ refer to the i^{th} city in the current partial sequence. The steps involved are:

STEP 0. Initialize, $N = 1, S_p = \{1\}, S_a = \{2,..,N\}$ .For $j = 2,...,N. r(j) = 1$
STEP 1. Select new city. Find $j^* = \arg\min_{j \in S_a} \{c_{j,c(j)},\ or\ c_{c(j),j}\}$ .Set $n = n+1$

STEP 2. Insert j^*, update $r(j)$ $S_a = S_a - j^*$. Find City $i^* \in S_p$ such that $i^* = \arg\min_{[i] \in S_p} \{c_{[i]j^*} + c_{j^*,[i+1]} - c_{[i],[i+1]}\}$. Update $S_p = \{s_1,..,i^*,j^*,i^*+1,..,s_n\}$. For all $j \in S_a$ if $\min\{c_{j,j^*}, c_{j^*,j}\} < c_{j,r(1)}$ then $r(j) = j^*$. If $n < N$, go to 2.

As can be seen, the closest insertion algorithms a constructive method. In order to understand the steps involved, let us consider an example related to changeover times for a flexible manufacturing cell (FMC).

Example 2.1

Table 2.1 shows the changeover times for a flexible manufacturing-cell. A machine is finishing producing batch T1 and other batches are yet to be completed. We are to use the closest insertion heuristic to find a job sequence, treating the problem as a TSP.

Table 2.1. Changeover times (hrs)

From/To	T1	T2	T3	T4	T5
T1	-	8	14	10	12
T2	4	-	15	11	13
T3	12	17	-	1	3
T4	12	17	5	-	3
T5	13	18	15	2	-

Solution

Step 0 $S_p = \{1\}, S_a = \{2,3,4,5\}, r_{j \in S_a}(j) = 1; j = 2,..,5$ This is equivalent to choosing the first city from the partial-list and eliminating this city form the available list.

Step 1 Select the new city: find $j^* = \arg\min_{j \in S_a} \{c_{j,r[j]}\}$ and set $n = n+1$
$\min_{j \in S_a} \{c_{12}, c_{13}, c_{14}, c_{15}, c_{21}, c_{31}, c_{41}, c_{51}\} = \min\{8,14,10,12,0,0,0,0\} = 8$; $j^* =$ 2. But ignore c_{21}, c_{31}, c_{41} and c_{51} because city 1 is already considered in S_a.

Step 2 Insert city 2, and update $r(j)$ for the remaining jobs 3, 4, and 5 $S_p = \{1,2\}; S_a = \{3,4,5\}, c_{12} + c_{21} - c_{14} = 8 + 4 - 0 = 12$

Step 1 Select new city:
$\min\{c_{23}, c_{24}, c_{25}, c_{32}, c_{42}, c_{52}\} = \{15,11,13,17,17,18\} = 11$; $j^* = 4; n = 3$. So we have job 4 after job 1 or 2.

Step 2 Insert job 4 There are the following possibilities from $\{1,2\}$: $\{1,2,4\}$ or $\{1,4,2\}$
For $\{1,2,4\}, c_{12} + c_{24} - c_{14} = 8 + 11 - 10 = 9$
For $\{1,4,2\}, c_{14} + c_{42} - c_{12} = 10 + 17 - 8 = 19$
The minimum occurs for inserting job 2 after job 4. Update $r(j)$ for remaining jobs 3, 5 i.e., r(3) = r(5) = 4

Step 1 Select new job.
$\min_{j \in S_a}\{c_{34}, c_{54}, c_{43}, c_{45}\} = \min\{1,2,5,3\}; j^* = 3$
$\min = c_{34}$ but 4 is already considered. Hence, $j^* = 3$.
Step 2 Insert job 3
There are the following possibilities from $\{1,2,4\}$:
$\{1,2,4,3\} : c_{43} + c_{31} - c_{41} = 5 + 12 - 12 = 5$
$\{1,2,3,4\} : c_{23} + c_{34} - c_{24} = 15 + 1 - 11 = 5$
$\{1,3,2,4\} : c_{13} + c_{32} - c_{12} = 14 + 17 - 8 = 25$
Choosing $\{1,2,4,3\}$ breaks the tie. Updating $r[5] = 3$

Step 1 Select new job.
Since job 5 remains, $j^* = 5$
Step 2 Insert job 5
There are the following possibilities from $\{1,2,4,3\}$
$\{1,2,4,3,5\} : c_{35} + c_{51} - c_{31} = 3 + 13 - 12 = 4$
$\{1,2,3,4,3\} : c_{45} + c_{53} - c_{43} = 3 + 15 - 5 = 13$
$\{1,3,2,4,3\} : c_{25} + c_{54} - c_{24} = 13 + 2 - 11 = 4$
$\{1,5,2,4,3\} : c_{15} + c_{52} - c_{12} = 12 + 18 - 8 = 22$
Choosing $\{1,2,4,3,5\}$ breaks the tie as the final sequence. The cost = $c_{12} + c_{24} + c_{43} + c_{35} = 8 + 11 + 5 + 3 = 27$

The TSP construction is shown in Fig 2.3. The meaning of this solution is that batch *1* is first produced, followed by batch *2*, then batch *4*, then batch *3*, and finally batch *5*.

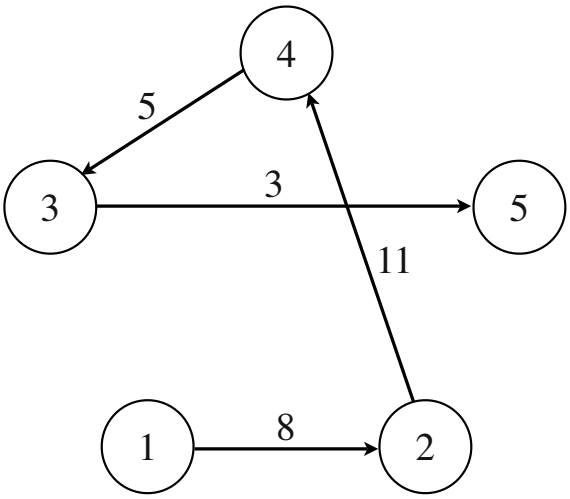

Fig. 2.3. TSP solution for Example 2.1

2.2.3 Automated Drilling Location and Hit Sequencing

Consider an automated drilling machine that drills holes on a flat sheet. The turret has to be loaded with all the tools required to hit the holes. There is no removing or adding of tools. The machine bed carries the flat plate and moves from home, locating each scheduled hit on the flat plate under the machine turret. Then the turret rotates and aligns the proper tool under the hit. This process continues until all hits are completed and the bed returns to the home position.

There are two problems to be solved here. One is to load tools to the turrets and the other is to locate or sequence hits. The objective is to minimize the cycle time such that the appropriate tools are loaded and the best hits-sequence is obtained. The problem can therefore be divided into two: (i) solve a TSP for the inter−hit sequencing; (ii) solve a quadratic assignment problem (QAP) for the tool loading. [21] developed a mathematical formulation to this problem and iterated between the TSP and QAP. Once the hit sequence is known, the sequence of tools to be used is then fixed since each hit requires specific tool. On the other hand, if we know the tool assignment on the turret, we need to know the inter-hit sequence. Connecting each hit in the best sequence is definitely a TSP, where we consider the machine bed home as the home for the TSP, and each hit, a city. Inter−hit travel times and the rotation of the turret are the costs involved and we take the maximum between them, i.e. inter−hit cost = max (inter−hit travel time, turret rotation travel time). The cost to place tool k in position i and tool l in position j is the time it takes the turret to rotate from i to j multiplied by the number of times the turret switches from tool k to l.

The inter-hit travel times are easy to estimate from the geometry of the plate to be punched and the tools required per punch. The inter−hits times are first estimated and then adjusted according to the turret rotation times. This information constitutes the data for solving the TSP. Once the hit sequence is obtained from the TSP, the tools are placed, by solving the QAP. Let us illustrate the TSP−QAP solution procedure by considering an example.

Example 2.2

A numerically controlled (NC) machine is to punch holes on a flat metal sheet and the hits are shown in Fig 2.4. The inter-hit times are shown in Table 2.2. There are four tools $\{a,b,c,d\}$ and the hits are $\{1,2,3,4,5,6,7\}$. The machine turret can hold five tools and rotates in clockwise or anti-clockwise direction. When the turret rotates from one tool position to an adjacent position, it takes *60* time units. It takes *75* time units and *90* time units to two locations and three locations respectively. The machine bed home is marked *0*. Assign tools to the turret and sequence the hits.

Solution

From the given inter−hit times, modified inter-hit times have to be calculated using the condition: inter−hit cost = max (inter-hit travel time, turret rotation travel time). For example, for inter−hit between locations *1* and *2*, the inter-hit travel time is *50* time units. Now, the tool for hit *1* is *c* while the tool for hit *2* is *a*. This means there is

Table 2.2. Inter−hit travel times

	Hit							
Hit	0	1	2	3	4	5	6	7
0	-	50	100	50	100	150	100	200
1	50	-	50	100	50	100	150	150
2	100	50	-	150	100	50	200	100
3	50	100	150	-	50	100	50	150
4	100	50	100	50	-	50	100	100
5	150	100	50	100	50	-	150	50
6	100	150	200	50	100	150	-	100
7	200	150	100	150	100	50	100	-

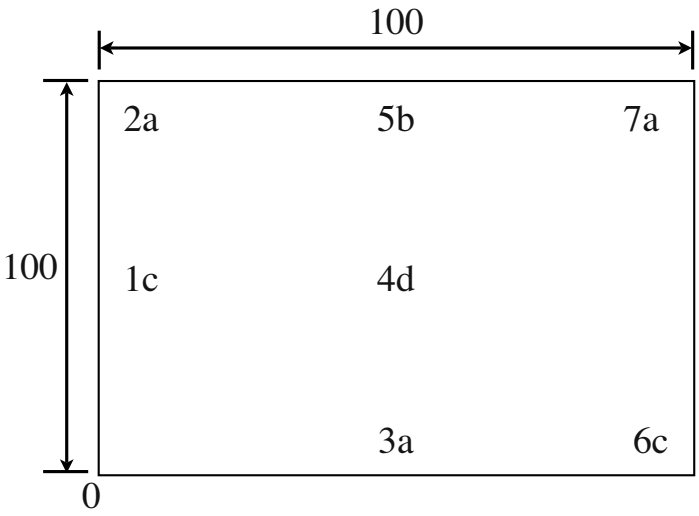

Fig. 2.4. Flat metal sheet to be punched

change in tools because that the turret will rotate. The cost of rotation is *60* time units, which exceeds the *50* inter-hit time unit. This means that the modified inter−hit time between locations *1* and *2* is *60* time units. From the home to any hit is not affected. The modified inter−hit times are shown in Table 2.3. This information is used for TSP. One TSP solution for Table 2.2 is $\{0,1,2,3,4,5,6,7\}$, with a cost of *830*. We used the DE heuristic to obtain tool sequence of $c \rightarrow d \rightarrow b \rightarrow a \rightarrow c \rightarrow a$, and the cost is *410*. As can be seen a better solution is obtain by the latter. Let us explain how we obtained the tool sequence. Solving the TSP using DE, the sequence obtaineed is $\{2,5,6,8,7,4,1,3\}$ or $\{1,4,5,7,6,3,0,2\}$. What we do is to refer to Fig 2.4 and get the labels that corrspond to this sequence as $\{c,d,b,a,c,a,a\}$. Hence the optimum sequence is $c-d-b-a-c$.

Table 2.3. Modified inter−hit travel times (considering turret movements)

	Hit							
Hit	0	1	2	3	4	5	6	7
0	-	50	100	50	100	150	100	200
1	50	-	60	100	60	100	150	150
2	100	60	-	150	100	60	200	100
3	50	100	150	-	60	100	60	150
4	100	60	100	60	-	60	100	100
5	150	100	60	100	60	-	150	60
6	100	150	200	60	100	150	-	100
7	200	150	100	150	100	60	100	-

2.2.4 Dynamic Pick and Place (DPP) Model of Placement Sequence and Magazine Assignment

Products assembled by robots are typical in present manufacturing system. To satisfy growing large scale demand of products efficient methods of product assemble is essential to reduce time frame and maximize profit. The Dynamic Pick and Place (DPP) model of Placement Sequence and magazine Assignment (SMA) is an interesting problem that could be solved using standard optimizing techniques, such as discrete or permutative DE. DPP model is a system consists of robot, assemble board and magazine feeder which move together with different speeds and directions depends on relative distances between assemble points and also on relative distances between magazine components. Major difficulty to solve this problem is that the feeder assignment depends on assembly sequence and vice versa. Placement sequence and magazine assignment (SMA) system has three major components robot, assembly board and component slots. Robot picks components from horizontal moving magazine and places into the predefined positions in the horizontal moving assembly board. To optimize production time frame assembly sequence and feeder assignment need to be determined. There are two models for this problem: Fixed Pick and place model and Dynamic Pick and Place model. In the FPP model, the magazine moves in x direction only while the board moves in both x-y directions and the robot arm moves between fixed "pick" and "place" points. In the DPP model, both magazine and board moves along x-axis while the robot arm moves between dynamic "pick" and "place" points. See Fig 2.5. Principal objective is to minimize total tardiness of robot movement hence minimize total assembly time.

There are few researchers who had solved the assembly sequence and feeder assignment problem by the DPP model. This is because this problem is quite challenging. [12] had proved that DPP has eliminated the robot waiting time by the FPP model. [11] used simulated annealing algorithm and obtained solutions better than previous approaches but the computation efficiency was quite low. Wang et al. have developed their own heuristic approach to come up with some good solutions. [19] proposed a new heuristic to improve Wang′s approach based on the fact that assembly time depends on the relative position of picking points as well as placement points. The main objective of DPP model is to eliminate the robot waiting time. To avoid tardiness robot arm tends to

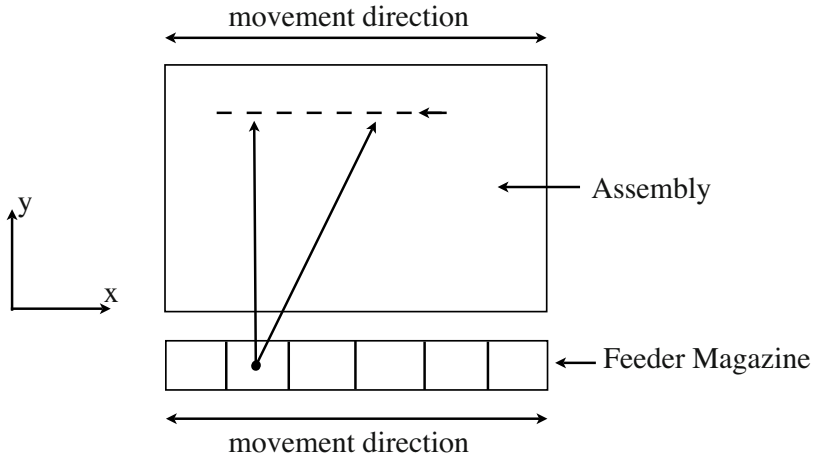

Fig. 2.5. Robot movement

move in shortest possible path (i.e. always tries to move in vertical direction) if vertical movement is not possible than it needs to stretch/compress its arm to avoid tardiness. This section reveals the formulation of DPP problem statement using the following notations:

V_a	speed of assembly arm
V_b	speed of board
$V\neg_m$	speed of magazine
N	number of placement components
K	number of component types ($K \leq N$)
$b(i)$	i^{th} placement in a placement sequence
$m(i)$	i^{th} placement in a pick sequence
$x^m_{i+1} = M^1_{i+1} + M^2_{i+1}$	interception distance of robot arm and magzine
$x^b_{i+1} = B^1_{i+1} + B^2_{i+1}$	interception distance of robot arm and board
$T(m(i), b(i))$	robot arm travel time from magazine location $m(i)$ to board location $m(i)$.
$T(b(i), b(i))$	robot arm travel time from board location $b(i)$ to magazine location $m(i)$.
$T place$	time taken to place the component
$T pick$	time taken to pick the component
CT	total assembly time

Fig 2.6 shows possible movements of board and magazine in DPP model [19]. Suppose the robot arm has finished placing the i^{th} component at point *B(i)* then moves to pick the next $(i+1)^{th}$ component from slot *M(i+1)* on the magazine. If magazine is able to travel distance $d(a,c) = ||a-c||$ before the robot actually arrives vertically towards

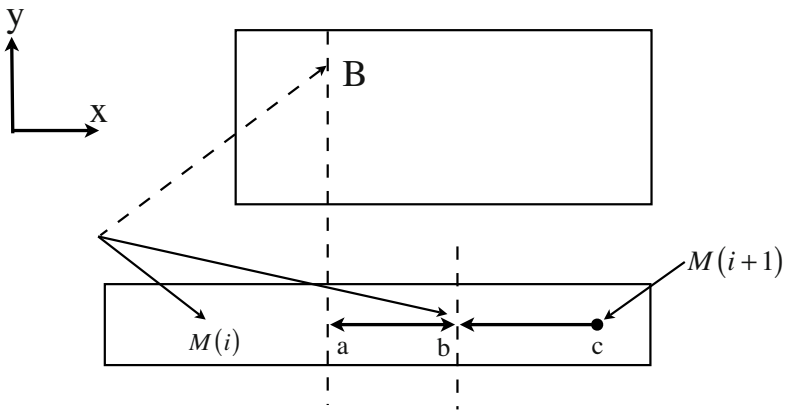

Fig. 2.6. DPP model

magazine then no interception will occur, but if the magazine fails then robot has to compress/stretch (intercept) its arm in x-direction of distance $d(a,b) = ||a-b||$ to get the component without waiting for the component *M(i+1)* to arrive at point a.

Fig 2.6 shows that slot $M(i+1)$ at point c has to reach at point a through travelling distance $d(a,c) = ||a-c||$ to avoid robot interception. Suppose magazine only managed to travel distance $d(b,c) = ||b-c||$ before the robot reached to the magazine i.e. $T(M(i),b(i)) + T_{place} + \frac{y_i}{v_r} \geq \frac{d(a,c)}{v_m}$. Then the robot has to stretch its arm to get to the point b. Hence tardiness is eliminated by robot interception. Exactly same movement principles are applied when robot arm moves from magazine to board.

Where total assembly time CT need to be minimized subject to constraints:

1. Board to magazine

$$
\begin{aligned}
&f\,(T(m(i),b(i)) + T_{place} + \tfrac{y_i}{v_r} \geq \tfrac{d(a,c)}{v_m})\ Then\\
&\qquad T(b(i),m(i+1)) = \tfrac{y_i}{v_m}\\
&else\\
&\qquad T(b(i),m(i+1) = \frac{\sqrt{(y_i^2+(x_{i+1}^m)^2)}}{v_r}\\
&Endif
\end{aligned}
$$

2. Magazine to board

$$
\begin{aligned}
&if\,(T(b(i),m(i+1)) + T_{pick} + \tfrac{y_i}{v_r} \geq \tfrac{d(a,c)}{vb})\ Then\\
&\qquad T(m(i+1),b(i+1)) = \tfrac{y_i}{v_m}\\
&else\\
&\qquad T(m(i+1),b(i+1) = \frac{\sqrt{(y_{i+1}^2+(x_{i+1}^b)^2)}}{v_r}\\
&Endif
\end{aligned}
$$

The formulation of this problem shows that DPP model is a function of i^{th} placement in a sequence $b(i)$, and i^{th} component in a pick sequence, $m(i)$. This is obviously a permutative-based combinatorial optimization problem which is challenging to solve.

2.2.5 Vehicle Routing Problem

Vehicle routing problem is delivery of goods to customers by a vehicle from a depot (see Fig 2.7). The goal here is to minimize the travelling distance and hence save cost. Here too an objective function would be created and inserted into the optimizer in order to obtain the best travelling path for which the cost is minimized. The application of vehicle routing problem can be applied in many places. One example is bin−picking problem. In some countries, the City Council bears a lot of extra costs on bin-picking vehicle by not following shortest path.

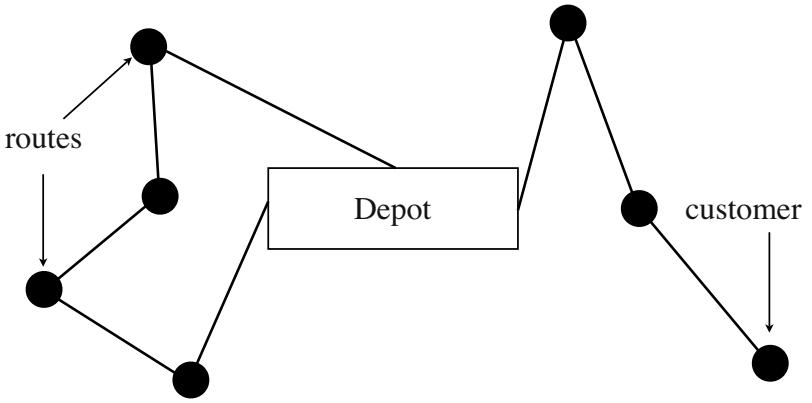

Fig. 2.7. DPP model

The CVRP is described as follows: n customers must be served from a unique depot. Each customer asks for a quantity for quantity q_i (where $i = 1,2,3,...,n$) of goods and a vehicle of capacity Q is available for delivery. Since the vehicle capacity is limited, the vehicle has to periodically return to the depot for reloading. Total tour demand is at most Q (which is vehicle capacity) and a customer should be visited only once [5].

2.2.6 Facility Location Problem

In facility location problem, we are given n potential facility location and m customers that must be served from these locations. There is a fixed cost c_j of opening facility j. There is a cost d_{ij} associated with serving customer i from facility j. We then have two sets of binary variables which are y_j is *1* if facility j is opened, *0* otherwise x_{ij} is *1* if customer i is served by facility j, *0* otherwise.

Mathematically the facility location problem can be formulated as

$$\begin{aligned} &\min \sum_{j=1}^{n} c_j y_j + \sum_{i=1}^{m} \sum_{j=1}^{n} d_{ij} x_{ij} \\ &s.t. \sum_{j=1}^{n} x_{ij} = K \quad \forall i \\ &x_{ij} \leq y_j \quad \forall i, j \\ &x_{ij}, y_j \in \{0, i\} \, \forall i, j \end{aligned} \tag{2.7}$$

2.3 Permutation-Based Combinatorial Approaches

This section describes two permutation−based combinatorial DE approaches which were merely described in [10] and three other permutation−based combinatorial DE approaches which are detailed in this book.

2.3.1 The Permutation Matrix Approach

The permutation matrix approach is the idea of Price, but Storn did the experiments that document its performance [10]. The permutative matrix approach is based on the idea of finding a permutative matrix that relates two vectors. For example, given two vectors x_{r1} and x_{r1} defined in Equation 2.8:

$$x_{r1} = \begin{pmatrix} 1 \\ 3 \\ 4 \\ 5 \\ 2 \end{pmatrix}, \quad x_{r2} = \begin{pmatrix} 1 \\ 4 \\ 3 \\ 5 \\ 2 \end{pmatrix}; \tag{2.8}$$

These two vectors encode tours, each of which is a permutation. The *permutation matrix,* $\boldsymbol{P}$, that x_{r1} and x_{r1} is defined as:

$$x_{r2} = P.x_{r1}, \; with \, P = \begin{pmatrix} 1\,0\,0\,0\,0 \\ 0\,0\,1\,0\,0 \\ 0\,1\,0\,0\,0 \\ 0\,0\,0\,1\,0 \\ 0\,0\,0\,0\,1 \end{pmatrix} \tag{2.9}$$

```
for(i = 1;i < M;i++) //search all columns of P
 {
  if(elementp(i,i) of P is 0) // 1 not on diagonal
   {
    if(rand() >δdel) //if random number ex [0,1] exceeds δdel
    {
     j = 1; // find row where p(j,i) = 1
     while(p(j,i)! = 1)j++;
    }
   }
 }
```

Fig. 2.8. Algorithm to apply the factor δ to the difference permutation, **P**

Price gives an algorithm that scales the effect of the permutation matrix as shown in Fig 2.8.

2.3.2 Adjacency Matrix Approach

Storn developed the adjacency matrix approach outlined in this section [10]. There are some rules that govern the adjacency matrix approach. When tours are encoded as city vectors, the difference between rotated but otherwise identical tours is never zero. Rotation, however, has no effect on a tour′s representation if it is encoded as an adjacency matrix. Storn defined the notation

$$(x+y) \mod 2 = x \oplus y \tag{2.10}$$

which is shorthand for modulo 2 addition, also known as the "exclusive or" logical operation for the operation of the matrices. The difference matrix Δ_{ij},

$$\Delta_{ij} = A_i \oplus A_j \tag{2.11}$$

is defined as the analog of DE′s traditional difference vector. Consider for example, the valid TSP matrices A_1 and A_2,

$$A_1 = \begin{pmatrix} 0\,1\,0\,0\,1 \\ 1\,0\,0\,1\,0 \\ 0\,0\,0\,1\,1 \\ 0\,1\,1\,0\,0 \\ 1\,0\,1\,0\,0 \end{pmatrix}, \qquad A_2 = \begin{pmatrix} 0\,1\,0\,0\,1 \\ 1\,0\,1\,0\,0 \\ 0\,1\,0\,1\,0 \\ 0\,0\,1\,0\,1 \\ 1\,0\,0\,1\,0 \end{pmatrix}, \tag{2.12}$$

and their difference given as

$$\Delta_{1,2} = A_1 \oplus A_2 = \begin{pmatrix} 0\,0\,0\,0\,0 \\ 0\,0\,1\,1\,0 \\ 0\,1\,0\,0\,1 \\ 0\,1\,0\,0\,1 \\ 0\,0\,1\,1\,0 \end{pmatrix} \tag{2.13}$$

From the definition of A_1 there are $1'$s in column 1 in rows $\{2 \text{ and } 5\}$, in column 2 there are $1'$s in rows $\{1 \text{ and } 4\}$, in column 3 there are $1'$s in rows $\{4 \text{ and } 5\}$, in column 4 there are $1'$s in rows $\{2 \text{ and } 3\}$, in column 5 there are $1'$s in rows $\{1 \text{ and } 3\}$ respectively. These pair−wise numbers define the adjacency relationships. Considering $\{2 \text{ and } 5\}$ and $\{4 \text{ and } 5\}$ it is shown that $'5'$ is common and $\{2 \text{ and } 4\}$ are adjacent. Considering $\{1 \text{ and } 4\}$ and $\{1 \text{ and } 3\}$ it is shown that $'1'$ is common and $\{1 \text{ and } 3\}$ are adjacent. Continuing in this manner it could be observed that Fig 2.9 shows the graphical interpretation of A_1, A_2 and Δ_{ij}.

2.3.3 Relative Position Indexing

In the relative position indexing approach [4], permutations are obtained by determining the relative sizes of the different parameters defining an instance. Let there be an

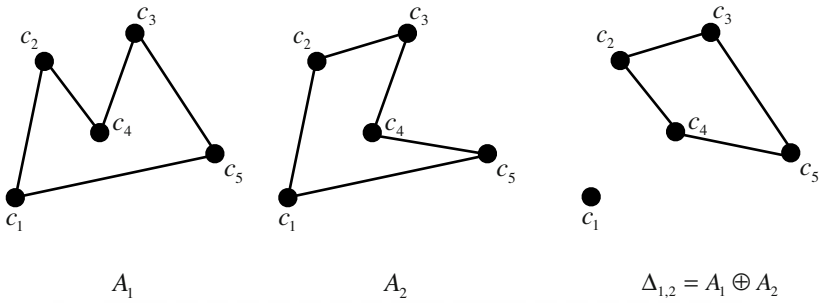

Fig. 2.9. Graphical interpretations of A_1, A_2 and the difference matrix Δ_{ij}

instance of four cities that define a tour such that these are initially generated by DE as $x_{1,f} = \{0.5\ 0.8\ 0.2\ 0.6\}$. Another instance of such four cities could simple be simply defined as $x_{2,f} = \{0.6\ 0.1\ 0.3\ 0.4\}$. In relative indexing, these instances encode permutations given as $x_1 = \{2\ 4\ 1\ 3\}$ and $x_2 = \{4\ 1\ 2\ 3\}$ respectively. In the first case for example the lowest value which is 0.2 is in the third position so it is allocated a label of 1; the next higher value is 0.5 which occupies the first position and it is allocated the label 2 and so on. Let there be a third instance denoted as $x_{3,f} = \{0.6\ 0.8\ 0.3\ 0.5\}$. Then we have $x_3 = \{3\ 4\ 1\ 2\}$. The concept is fairly simple. The subscript f indicates floating point.

The basic idea behind DE is that two vectors define a difference that can then be added to another vector as a mutation. The same idea transfers directly to the realm of permutations, or the permutation group. Just as two vectors in real space define a difference vector that is also a vector, two permutations define a mapping that is also a permutation. Therefore, when mutation is applied to with F = 0.5, the floating-point mutant vector, v_f, is

$$\begin{aligned} v_f &= x_{r3,f} + F\left(x_{r1,f} - x_{r2,f}\right) \\ &= \{0.6\ 0.8\ 0.3\ 0.5\} + 0.5\{-0.1\ 0.7\ -0.1\ 0.2\} \\ &= \{0.55\ 1.15\ 0.25\ 0.6\} \end{aligned} \tag{2.14}$$

The floating-point mutant vector, v_f, is then transformed back into the integer domain by assigning the smallest floating value (0.25) to the smallest integer (1), the next highest floating value (0.55) to the next highest integer (2), and so on to obtain $v = \{2\ 4\ 1\ 3\}$. [10] noted that this backward transformation, or "relative position indexing", always yields a valid tour except in the unlikely event that two or more floating−point values are the same. When such an event occurs, the trial vector must be either discarded or repaired.

2.3.4 Forward/Backward Transformation Approach

The forward/backward transformation approach is the idea of [6], and is generally referred to as Onwubolu′s approach [10]. There are two steps involved:

Forward Transformation

The transformation scheme represents the most integral part of the code. [6] developed an effective routine for the conversion of permutative-based indices into the continuous domain. Let a set of integer numbers be represented as $\mathbf{x}_i \in \mathbf{x}_{i,G}$ which belong to solution $x_{j,i,G=0}$. The formulation of the forward transformation is given as:

$$\mathbf{x}'_i = -1 + \alpha \mathbf{x}_i \tag{2.15}$$

where the value α is a small number.

Backward Transformation

The reverse operation to forward transformation, converts the real value back into integer as given in 2.16 assuming $\mathbf{x}'$ to be the real value obtained from 2.15.

$$\text{int}[\mathbf{x}_i] = \left(1 + \mathbf{x}'_i\right)/\alpha \tag{2.16}$$

The value $\mathbf{x}_i$ is rounded to the nearest integer. [9], [2, 3] have applied this method to an enhanced DE for floor shop problems.

2.3.5 Smallest Position Value Approach

The smallest position value (SPV) approach is the idea of [20] in which a unique solution representation of a continuous DE problem formulation is presented and the SPV rule is used to determine the permutations. Applying this concept to the GTSP, in which a tour is required, integer parts of the parameter values (s_j) in a continuous DE problem formulation represent the nodes (v_j). Then the random key values (s_j) are determined by simply subtracting the integer part of the parameter x_j from its current value considering the negative signs, i.e., $s_j = x_j - \text{int}(x_j)$. Finally, with respect to the random key values (s_j) , the smallest position value (SPV) rule of [20] is applied to the random key vector to determine the tour π. They adapted the encoding concept of [1] for solving the GTSP using GA approach, where each set V_j has a gene consisting of an integer part between $\left[1, \left|V_j\right|\right]$ and a fractional part between $[0,1]$. The integer part indicates which node from the cluster is included in the tour, and the nodes are sorted by their fractional part to indicate the order. The objective function value implied by a solution x with m nodes is the total tour length, which is given by

$$F(\pi) = \sum_{j=1}^{m-1} d_{\pi_j \pi_{j+1}} + d_{\pi_m \pi_1} \tag{2.17}$$

$V = \{1,..,20\}$ and $V_1 = \{1,..,5\}$, $V_2 = \{6,,..,10\}$, $V_3 = \{11,..,15\}$ and $V_4 = \{16,..,20\}$. Table 2.4 shows the solution representation of the DE for the GTSP.

In Table 2.4, noting that $\left[1, \left|V_j\right|\right]$, the integer parts of the parameter values (s_j) are respectively decoded as $\{4,3,1,3\}$. These decoded values are used to extract the nodes from the clusters V_1, V_2, V_3, V_4. The first node occupies the fourth position in V_1, the second node occupies the third position in V_2, the third node occupies the first position in

Table 2.4. SPV Solution Representation

j	1	2	3	4
x_j	4.23	-3.07	1.80	3.76
v_j	4	8	11	18
s_j	0.23	-0.07	0.80	0.76
π_j	8	4	18	11
$F(\pi)$	$d_{8,4}$	$d_{4,18}$	$d_{18,11}$	$d_{11,8}$

V_3, while the fourth node occupies the third position in V_4. Extracting these labels show that the nodes are $\{4,8,11,18\}$ The random key values are $\{0.23,-0.07,0.80,0.76\}$; finally, with respect to the random key values (s_j), the smallest position value (SPV) rule is applied to the random key vector by arranging the values in a non-descending order $\{-0.07,0.23,0.76,0.08\}$ to determine the tour $\pi\{8,4,18,11\}$. Using equation 2.17, the total tour length is then obtained as

$$F(\pi) = \sum_{j=1}^{m-1} d_{\pi_j \pi_{j+1}} + d_{\pi_m \pi_1} = d_{8,4} + d_{4,18} + d_{18,11} + d_{11,8}$$

In this approach, a problem may rise such that when the DE update equations are applied, any parameter value might be outside of the initial search range, which is restricted to the size of each cluster. Let $x_{\min}[j]$ and $x_{\max}[j]$ represent the minimum and maximum value of each parameter value for dimension j. Then they stand for the minimum and maximum cluster sizes of each dimension j. Regarding the initial population, each parameter value for the set V_j is drawn uniformly from $[-V_j+1, V_j+1]$. Obviously, $x_{\max}[j]$ is restricted to $[V_j+1]$, whereas $x_{\min}[j]$ is restricted to $-x_{\max}[j]$. During the reproduction of the DE, when any parameter value is outside of the cluster size, it is randomly reassigned to the corresponding cluster size again.

2.3.6 Discrete/Binary Approach

Tasgetiren et al. present for the first time in this chapter, the application of the DDE algorithm to the GTSP. They construct a unique solution representation including both cluster and tour information is presented, which handles the GTSP properly when carrying out the DDE operations. The Population individuals can be constructed in such a way that first a permutation of clusters is determined randomly, and then since each cluster contains one or more nodes, a tour is established by randomly choosing a single node from each corresponding cluster. For example, n_j stands for the cluster in the j^{th} dimension, whereas π_j represents the node to be visited from the cluster n_j.

Now, consider a GTSP instance with $N = \{1,..,25\}$ where the clusters are $n_1 = \{1,..,5\}$, $n_2 = \{6,..,10\}$, $n_3 = \{11,..,15\}$, $n_4 = \{16,..,20\}$ and $n_5 = \{21,..,25\}$. Table 2.5 shows the discrete/binary solution representation of the DDE for the GTSP.

A permutation of clusters is determined randomly as $\{4,1,5,2,3\}$. This means that the first node is randomly chosen from the fourth cluster (here 16 is randomly chosen); the second node is randomly chosen from the first cluster (here 5 is randomly chosen);

Table 2.5. Discrete/binary Solution Representation

	j	1	2	3	4	5
	n_j	4	1	5	2	3
X	π_j	16	5	22	8	14
	$d_{\pi_j\pi_{j+1}}$	$d_{16,5}$	$d_{5,22}$	$d_{22,8}$	$d_{8,14}$	$d_{14,16}$

the third node is randomly chosen from the fifth cluster (here 22 is randomly chosen); the fourth node is randomly chosen from the second cluster (here 8 is randomly chosen); and the fifth node is randomly chosen from the third cluster (here 14 is randomly chosen).

As already illustrated, the objective function value implied by a solution x with m nodes is the total tour length, which is given by:

$$F(\pi) = \sum_{j=1}^{m-1} d_{\pi_j\pi_{j+1}} + d_{\pi_m\pi_1} \tag{2.18}$$

This leads to the total tour length being obtained as

$$F(\pi) = \sum_{j=1}^{m-1} d_{\pi_j\pi_{j+1}} + d_{\pi_m\pi_1} = d_{16,5} + d_{5,22} + d_{22,8} + d_{8,14} + d_{14,16}$$

2.3.7 Discrete Set Handling Approach

Discrete set handling is an algorithmic approach how to handle in a numerical way objects from discrete set. Discrete set usually consist of various elements with non-numerical nature. In its canonical form DE is only capable of handling continuous variables. However extending it for optimization of discrete variables is rather easy. Only a couple of simple modifications are required. In evolution instead of the discrete value x_i itself, we may assign its index, i, to x. Now the discrete variable can be handled as an integer variable that is boundary constrained to range $< 1,2,3,.....,N >$. So as to evaluate the objective function, the discrete value, x_i, is used instead of its index i. In other words, instead of optimizing the value of the discrete variable directly, we optimize the value of its index i. Only during evaluation is the indicated discrete value used. Once the discrete problem has been converted into an integer one, the methods for handling integer variables can be applied. The principle of discrete parameter handling is depicted in chapter 7.3.

2.3.8 Anatomy of Some Approaches

[10] carried out anatomy of the four permutation−based combinatorial DE approaches described in their book (see Table 2.6). This exercise excludes smallest position value, discrete/binary and discrete set handling approaches.

Table 2.6. Anatomy of four permutation-based combinatorial DE approaches[+]

Approach	Observations
Permutation Matrix	In practice, this approach tends to stagnate because moves derived from the permutation matrix are seldom productive. In addition, this method is unable to distinguish rotated but otherwise equal tours. Because they display a unique binary signature, equal tours can be detected by other means, although this possibility is not exploited in the algorithm described in Fig 2.8.
Adjacency Matrix	(1) This scheme preserves good sections of the tour if the population has almost converged, i.e., if most of the TSP matrices in the population contain the same sub−tours. When the population is almost converged, there is a high probability that the difference matrix will contain just a few ones, which means that there are only a few cities available for a 2−exchange.
Relative Position Indexing	(1) This approach resembles traditional DE because they both use vector addition, although their ultimate effect is to shuffle values between parameters, i.e., generate permutations. (2) This approach impedes DE′s self-steering mechanism because it fails to recognize rotated tours as equal. (3) A closer look, however, reveals that DE′s mutation scheme together with the forward and backward transformations is, in essence, a shuffling generator. (4) In addition, this approach does not reliably detect identical tours because the difference in city indices has no real significance. For example, vectors with rotated entries, e.g., $(2,3,4,5,1)$ and $(1,2,3,4,5)$, are the same tour, but their difference, e.g., $(1,1,1,1,-4)$, is not zero.
Forward/backward Transformation	(1) This approach resembles traditional DE because they both use vector addition, although their ultimate effect is to shuffle values between parameters, i.e., generate permutations. (2) This approach impedes DE′s self−steering mechanism because it fails to recognize rotated tours as equal. (3) In addition, Onwubolu′s method usually generates invalid tours that must be repaired. Even though competitive results are reported in Onwubolu there is reason to believe that the success of this approach is primarily a consequence of prudently chosen local heuristics and repair mechanisms, not DE mutation.

[+] Described in Price et al. (2005).

2.4 Conclusions

There has been some reservation that although DE has performed well on wide−sense combinatorial problems, its suitability as a combinatorial optimizer is still a topic of considerable debate and a definitive judgment cannot be given at this time. Moreover, it is said that although the DE mutation concept extends to other groups, like the permutation group, there is no empirical evidence that such operators are particularly effective. The opinion expressed in this book is similar to that of [18] that all good heuristics are

able to transform a combinatorial problem into a space which is amenable for search, and that there is no such thing as an "all-cure" algorithm for combinatorial problems. For example, particle swarm optimization (PSO) works fairly well for combinatorial problems, but only in combination with a good tailored heuristic. If such a heuristic is used, then PSO can locate promising regions. The same logic applies to a number of optimization approaches.

While the anatomy described in Table 2.6 favors the adjacency matrix and permutation matrix approaches, compared to the forward/backward transformation relative position indexing approaches, it is not known in the literature where the adjacency matrix and permutation matrix approaches have been applied to real−life permutation-based combinatorial problems.

In this book, it is therefore concluded that:

1. The original classical DE which Storn and Price developed was designed to solve only problems characterized by continuous parameters. This means that only a subset of real-world problems could be solved by the original canonical DE.
2. For quite some time, this deficiency made DE not to be employed to a vast number of real-world problems which characterized by permutative-based combinatorial parameters.
3. This book complements that of [10] and vice versa. Taken together therefore, both books will be needed by practitioners and students interested in DE in order to have the full potentials of DE at their disposal. In other words, DE as an area of optimization is incomplete unless it can deal with real−life problems in the areas of continuous space as well as permutative-based combinatorial domain.

References

1. Bean, J.: Genetic algorithms and random keys for sequencing and optimization. ORSA, Journal on Computing 6, 154–160 (1994)
2. Davendra, D., Onwubolu, G.: Flow Shop Scheduling using Enhanced Differential Evolution. In: Proceeding of the 21st European Conference on Modelling and Simulation, Prague, Czech Republic, June 4-5, pp. 259–264 (2007)
3. Davendra, D., Onwubolu, G.: Enhanced Differential Evolution hybrid Scatter Search for Discrete Optimisation. In: Proceeding of the IEEE Congress on Evolutionary Computation, Singapore, September 25-28, pp. 1156–1162 (2007)
4. Lichtblau, D.: Discrete optimization using Mathematica. In: Callaos, N., Ebisuzaki, T., Starr, B., Abe, M., Lichtblau, D. (eds.) World multi−conference on systemics, cybernetics and informatics (SCI 2002), International Institute of Informatics and Systemics, vol. 16, pp. 169–174 (2002) (Cited September 1, 2008), http://library.wolfram.comlinfocenter/Conferences/4317
5. Mastolilli, M.: Vehicle routing problems (2008) (Cited September 1, 2008), http://www.idsia.ch/~monaldo/vrp.net
6. Onwubolu, G.: Optimisation using Differential Evolution Algorithm. Technical Report TR-2001-05, IAS (October 2001)
7. Onwubolu, G.: Optimizing CNC drilling machine operation: traveling salesman problem-differential evolution approach. In: Onwubolu, G., Babu, B. (eds.) New optimization techniques in engineering, pp. 537–564. Springer, Heidelberg (2004)

8. Onwubolu, G., Clerc, M.: Optimal path for automated drilling operations by a new heuristic approach using particle swarm optimization. Int. J. Prod. Res. 42(3), 473–491 (2004)
9. Onwubolu, G., Davendra, D.: Scheduling flow shops using differential evolution algorithm. Eur. J. Oper. Res. 171, 674–679 (2006)
10. Price, K., Storn, R., Lampinen, J.: Differential Evolution. Springer, Heidelberg (2005)
11. Su, C., Fu, H.: A simulated annealing heuristic for robotics assembly using the dynamic pick−and−place model. Prod. Plann. Contr. 9(8), 795–802 (1998)
12. Su, C., Fu, H., Ho, L.: A novel tabu search approach to find the best placement sequence and magazine assignment in dynamic robotics assembly. Prod. Plann. Contr. 9(6), 366–376 (1998)
13. Storn, R.: Differential evolution design of an R−filter with requirements for magnitude and group delay. In: IEEE international conference on evolutionary computation (ICEC 1996), pp. 268–273. IEEE Press, New York (1996)
14. Storn, R.: On the usage of differential evolution for function optimization. In: NAFIPS, Berkeley, pp. 519–523 (l996)
15. Storn, R.: System design by constraint adaptation and differential evolution. IEEE Trans. Evol. Comput. 3(I), 22–34 (1999)
16. Storn, R.: Designing digital filters with differential evolution. In: Come, D., et al. (eds.), pp. 109–125 (1999)
17. Storn, R.: (2000) (Cited September 1, 2008), `http://www.icsi.berkeley.edu/~stornlfiwiz.html`
18. Storn, R., Price, K.: Differential evolution–a simple and efficient heuristic for global optimization over continuous spaces. J. Global Optim. 11, 341–359 (1997)
19. Tabucanon, M., Hop, N.: Multiple criteria approach for solving feeder assignment and assembly sequencing problem in PCB assembly. Prod. Plann. Contr. 12(8), 736–744 (2001)
20. Tasgetiren, M., Sevkli, M., Liang, Y.-C., Gencyilmaz, G.: Particle Swarm Optimization Algorithm for the Single Machine Total Weighted Tardiness Problem. In: The Proceeding of the World Congress on Evolutionary Computation, CEC 2004, pp. 1412–1419 (2004)
21. Walas, R., Askin, R.: An algorithm for NC turret punch tool location and hit sequencing. IIE Transactions 16(3), 280–287 (1984)