VŠB – Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Telecommunications

Analýza dynamiky evolučních algoritmů pomocí komplexních sítí aplikovaných na kombinatorické optimalizační problémy

Complex Network Analysis of Evolutionary Algorithms Applied to Combinatorial Optimization Problem

2014 Jan Gazda

VŠB - Technical University of Ostrava
Faculty of Electrical Engineering and Computer Science
Department of Telecommunications

Bachelor Thesis Assignment

Student:	**Jan Gazda**
Study Programme:	B2647 Information and Communication Technology
Study Branch:	2601R013 Telecommunication Technology
Title:	Analýza dynamiky evolučních algoritmů pomocí komplexních sítí aplikovaných na kombinatorické optimalizační problémy Complex Network Analysis of Evolutionary Algorithms Applied to Combinatorial Optimization Problem.

Description:

This thesis requires the analysis of the application of evolutionary algorithms to the task of solving combinatorial optimization problems. The analysis will require generic complex network tools from which the evolutionary dynamics will be determined.

The main tasks of the thesis will include:

1. Development of an evolutionary algorithm, which will be used to solve a combinatorial optimization problem.
2. Development of a generation based complex network hierarchy in the evolutionary algorithm.
3. The analysis of the complex network using the following methodologies:
   a. Community graph
   b. Adjacency Graph
   c. Find K-Clan
   d. Degree of Centrality
   e. Closeness of Centrality
   f. Mean Neighbourhood Degree
   g. Core Components
   h. Lamda components.
4. The development of a corresponding CML system based on the analysis of the complex network.
5. An analysis of the linkage between the evolutionary algorithm dynamics and the CML system.

References:

1. M. Pinedo. 1995. Scheduling: theory, algorithms and systems. Prentice Hall, Inc., New Jersey.
2. R. Cohen and S. Havlin. 2010. Complex Networks: Structure, Robustness and Function, Cambridge University Press, UK. ISBN: 978-0521841566.
3. B. Korte and J. Vygen 2012. Combinatorial Optimization: Theory and Algorithms (Algorithms and Combinatorics), Springer, Germany, ISBN: 978-3642244872

Extent and terms of a thesis are specified in directions for its elaboration that are opened to the public on the web sites of the faculty.

Supervisor: **doc. MSc. Donald David Davendra, Ph.D.**

Date of issue: 01.09.2013

Date of submission: 07.05.2014

[illegible]

VYSOKÁ ŠKOLA BÁŇSKÁ-TECHNICKÁ UNIVERZITA
Fakulta elektrotechniky a informatiky
1
* OSTRAVA *

[illegible]

doc. Ing. Miroslav Vozňák, Ph.D.
Head of Department

prof. RNDr. Václav Snášel, CSc.
Dean of Faculty

I hereby declare that this bachelor's thesis was written by myself. I have quoted all the references I have drawn upon.

Ostrava May 7, 2014

..........[illegible]..........

My gratitude goes to **doc. MSc. Donald David Davendra, Ph.D.**, who gave me an opportunity to contribute to such an interesting project. I also thank him for his professional advice, leadership and last but not least for his motivating and encouraging personal approach.

Abstrakt

Tato práce se zabývá spojením mezi evolučními algoritmy (EA) a komplexnímí sítěmi (KS). EA jsou biologicky inspirované algoritmy napodobující přirozené přírodní jevy s cílem modelovat a řešit složité technické problémy. Jednou z jejich funkcí je snaha napodobit evoluční dogma. Chování celé populace je skrze její vývoj sledováno pomocí nástrojů pro analýzu komplexních sítí. Analyzovány jsou tyto čtyři atributy: matice sjednocení, centralita, kliky a komunity. Byla provedena řada experimentů a analýz, ze kterých pomocí těchto atributů, byly získány zajímavé informace týkající se vývoje populace, stagnace, propojení a hierarchie. Získaná data nastínila koncepci populační dynamiky a dala by se využít ke kontrole samotné evoluce.

Klíčová slova: Komplexní sítě, Evoluční Algoritmy, Diferenciální Evoluce, Diskrétní SamoOrganizující se Migrační algoritmus

Abstract

This thesis explores the connection between Evolutionary Algorithms (EA's) and Complex Networks (CN's). EA's are bio-inspired algorithms which mimic naturally occurring phenomena in order to model and solve complex engineering tasks. One of its features is its population based paradigm. The behaviour of the population over the iterations is analysed in this thesis using CN analysis tools. Four distinct broad attributes are analysed; adjacency matrix, centralities, cliques and communities. Using these attributes, a number of experimentations and analysis were conducted, from which interesting information regarding population development, stagnation, network interconnection and hierarchical development was obtained. These data supported the concept of population dynamics and furthermore could be used for population and evolution control.

Keywords: Complex Networks, Evolutionary Algorithms, Enhanced Differential Evolution, Discrete Self-Organising Migrating Algorithm

List of abbreviations and symbols

SOMA	–	Self Organizing Migrating Algorithm
DSOMA	–	Discrete Self Organizing Migrating Algorithm
DE	–	Differential Evolution Algorithm
EDE	–	Enhanced Differential Evolution Algorithm
CN	–	Complex Network
FSS	–	Flow shop scheduling
FSSB	–	Flow shop scheduling with blocking constraint
FSSNW	–	Flow shop scheduling with no-wait constraint

Contents

List of Tables

List of Figures

Introduction

The thesis is focused on the analysis of Evolutionary Algorithms (EA's), which are applied to optimise combinatorial optimisation problems. Generally, EA's can be classified as bio-inspired heuristics, which mimic nature in order to solve extremely complex problems, in the field of Engineering and Mathematics amongst others. Some of the most complex problems in Manufacturing Engineering as those of Scheduling, where the goal is to find the optimal schedule of tasks to machines in order to minimise time and operating cost. EA's use three generic attributes to accomplish this task; the use of a *population* (a group of possible solutions), *stochasticity* (application of randomness) and *generation* (evolution of the population over time).

This thesis analyses an EA in terms of its population, and how the population evolves over time, and what attributes exist within the population throughout the generations. One of the tools for such an analysis is the Complex Network (CN). Some of the aims of this thesis is to answer the following questions.

1. Does complex network form in EA's?
2. How do different EA's compare with each other in terms of CN?
3. Is it possible to obtain meaningful information using CN from an EA?

The work is divided in the following parts. The first chapter is a short introduction into CN, their generic types and models. I will also describe these attributes, which were used for the CN analysis:

a. Graph Representation of Networks

b. Average Path Length

c. Clustering Coefficient

d. Degree Distribution

e. Models of complex networks

A brief outline of the current state-of-art in the field of CN analysis of EA's is then presented, which is predominantly in the real-domain problem scope.

The second chapter introduces the EA's of Differential Evolution (DE) [2], Enhanced Differential Evolution (EDE) [3], Self Organising Migrating Algorithm (SOMA) [4] and Discrete Self Organising Migrating Algorithm (DSOMA) [5]. EDE and DSOMA are the discrete variants of DE and SOMA, which are used to solve the combinatorial optimisation problem.

The final chapter contains experiments with EDE and DSOMA applied on the combinatorial optimisation problems of flowshop, flowshop with blocking and flowshop with no-wait makespan.

The following CN attributes are analysed for each algorithm:

1. Adjacency Graph
2. Minimal Cut
3. Degree Centrality
4. Closeness Centrality
5. Betweenness Centrality
6. Katz Centrality
7. Mean Neighbourhood Degree
8. k-Clique
9. k-Plex
10. k-Club
11. k-Clan
12. Community Graph Plot

Using the above mentioned CN attributes, complex network analysis will be done for the different problems and conclusion drawn from the results.

1 Complex Networks and their Models

The description and characterisation of complex networks can be done by many different concepts and measures. These three fundamental measures will be used: average path length, clustering coefficient, and degree distribution.

1.1 Graph Representation of Networks

A graph is an algebraic structure that describes the objects and relations between them [6]. It can be imagined as a graph with few objects (vertices) and their connectors (edges). Even the network can be represented as a graph in the same way as the Eüler's problem the seven-bridges of the Königsburg (Fig. 1.1).

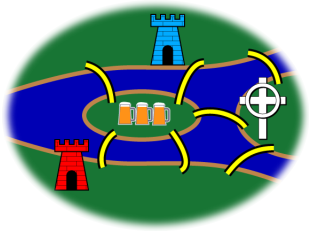


Figure 1.1: Seven Bridges of Königsberg

A given network can be presented by a set V of *nodes* and a set E of *edges*, connected together as a *graph* denoted as an ordered pair $G = (V, E)$ where the total number of nodes is $N = |V|$ and that of edges is $M = |E|$. Each edge $e \in E$ is connected to the pair of nodes one at each end.

This network is called an *undirected network* (Fig. 1.2a) if any pair of nodes (i, j) and (j, i) respectively in reverse ordering is connected by the same edge; otherwise the network is called a *directed network* (Fig. 1.2d).

The edges or nodes can be assigned by a weight value, where the network is called a *weighted network* (Fig. 1.2c); otherwise, it is an *unweighted network*. If all weight values are equal to each other, then the network can be also viewed as unweighted. The weight value is usually used for calculation of flow in the network or searching of minimum spanning tree etc. *Simple graph* can have at only one edge connecting two nodes and self-loops are forbidden. Self-loop is edge connecting node to itself. Otherwise it can be *multigraph or pseudograph* (1.2e) [6].

1.2 Average Path Length

The distance in networks can be measured as the number of edges connecting two nodes labeled i and j. But the term of path means total number of edges between two nodes that connect them through shortest linkages. This distance is denoted as d_{ij} which means distance from node i to node j. For example, Fig. 1.2a the distance from node D to node F could be 2 or 3 depends on the way you take to the desired node.

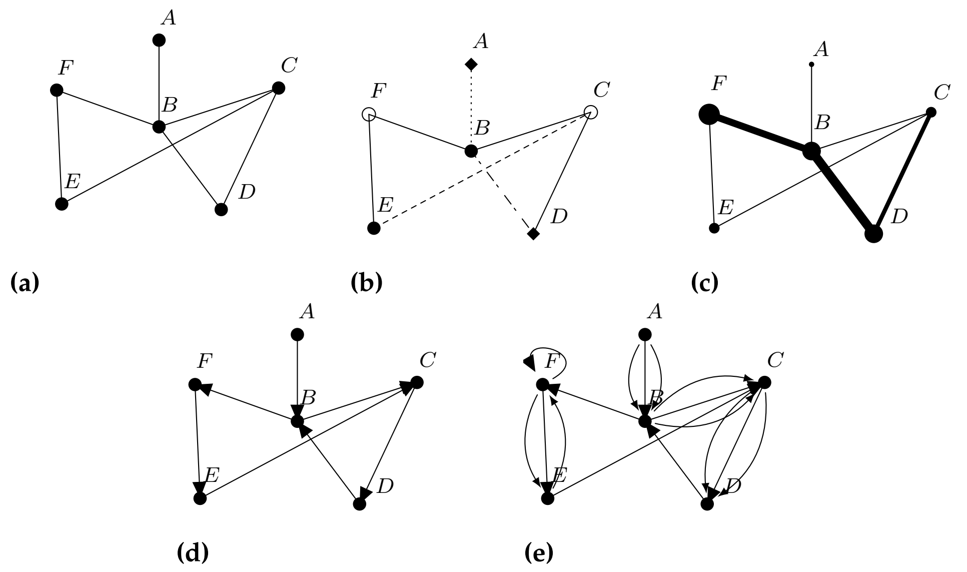


Figure 1.2: A few examples of different types of graphs:
(a) an undirected and un-weighted graph with same type of nodes
(b) an undirected and un-weighted graph with different types of nodes and edges
(c) an undirected but weighted graph with weights on both nodes and edges
(d) a directed but un-weighted graph with same type of nodes
(e) a directed multigraph with self-loop

The paths from D to node F are:

a) $D \to C \to B \to F = 3$

b) $D \to B \to F = 2$

The path a) goes through three edges and path b) only through two. So the distance is $d_{DF} = 2$. As you can see the path through shortest linkages between nodes D and F is 2. The diameter of a network, denoted D, is defined to be the largest of all distances in the network as in Equation 1.1 [6].

$$D = \max_{i,j} d_{ij} \tag{1.1}$$

This can be found on the path from the node A to the node E, because these nodes are connected through exactly 3 edges and its the only and shortest path that connecting them and this is also the largest of all distances. The set of all shortest paths is $S = \{d_{AB} = 1, d_{AC} = 2, d_{AD} = 2, \mathbf{d_{AE} = 3}, d_{AF} = 2, d_{BC} = 1, d_{BD} = 1, d_{BE} = 2, d_{BF} = 1, d_{CD} = 1, d_{CE} = 1, d_{CF} = 2, d_{DE} = 2, d_{D,F} = 2, d_{EF} = 1\}$ So the diameter of our example network is 3. The average path length of a network is defined to be the average value of all distances over the network as given in Equation 1.2.

$$L = \frac{2}{N(N-1)} \sum_{i<j} d_{ij} \tag{1.2}$$

Where N is the size of the network, the total number of nodes in the network. The self-loop on node F of network on Fig.1.2e is not considered as single edge, therefore is not counted in this formula. It is clear that the average path length of a network with N nodes and M edges can be searched by an optimisation algorithm with computational complexity $O(NM)$.

1.3 Clustering Coefficient

In a social network, two people may have common friend but they can even be friends themselves. If one node have two other neighbours, there is the probability that those two nodes are neighbours to each other. Generally, lets say that node i degree k_i have neighbours whose have other k_i neighbours there is a probability that these nodes are somehow connected together. So the all possible connections among all these nodes can be obtained as $k_i(k_i - 1)/2$. But what if the actual number of connections is different? Let E_i be the number of actual connections among these k_i nodes. Then the ratio between actual and all possible number of connections of the neighbours of node i is the local *clustering coefficient* of the node, denoted C_i (Equation 1.1) [6].

$$C_i = \frac{E}{\binom{k_i}{2}} = \frac{2E_i}{ki(k_i - 1)} \tag{1.3}$$

The global clustering coefficient of the network is the probability of the third connection between three randomly chosen nodes. Equation 1.4 gives an average number of all clustering coefficients for all nodes in the network [7].

$$\overline{C} = \frac{1}{N}\sum_{i=0}^{N} C_i \tag{1.4}$$

1.4 Degree Distribution

The degree of node i in an undirected network is the number k_i of the edges directly connected to the node. On the other hand in a directed network the degree of node can be the number of incoming or outgoing edges. More important nodes in the network usually have higher degree than less important nodes. The average value node degree in the network is given as an arithmetic mean of all node degrees in the network (Equation 1.5).

$$\langle k \rangle = \frac{1}{N}\sum_{i=1}^{N} k_i \tag{1.5}$$

The degree distribution function can give us a good overview as to how many nodes have degree lower than average and higher than average. Thus we can imagine how many hubs and stubs the network has and it show how the structure of the network will look like.

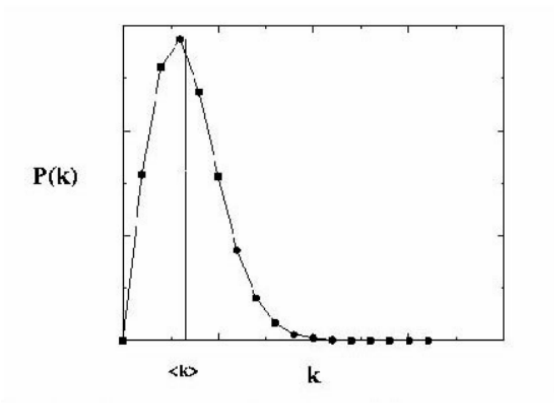
In a fully connected network, where all nodes have same degree every chosen node would have same degree. So the probability to chose random node that has degree k is the *degree distribution* as given in Equation 1.6 [6].

$$\begin{aligned} P(k) = &\text{ Probability that a node has degree } k \text{ ,} \\ &\text{ where the node is picked at random uniformly} \end{aligned} \tag{1.6}$$

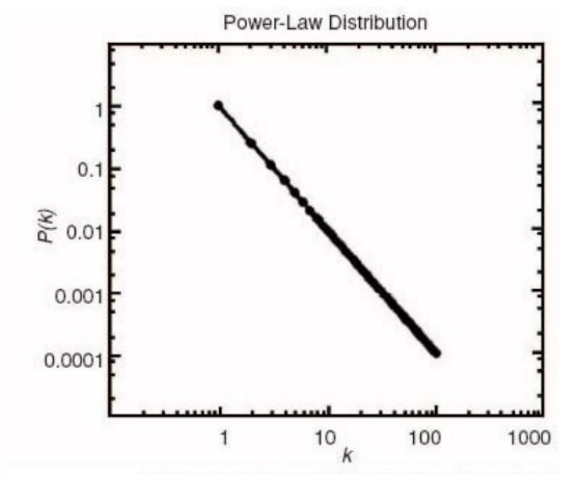
Two typical degree distributions of random and scale-free networks are shown on Fig. 1.3a. The *Poisson* distribution says that the network have a large number of nodes with same degree. The randomness of the network can be determined from the curve of Poisson distribution, the narrower the curve is the more regular the network is. The *Delta* distribution is the case when the curve become a line parallel to $P(k)$ axis, then all nodes in the network will have the same degree. Cases between Delta and Poisson distribution are described by the *power-law* function. The *power-law* function is described in Equation 1.7 [6].

$$P(k)\ k^{-\gamma} \tag{1.7}$$

The network that (at least) asymptotically follows the power-law distribution is called *scale-free*.



(a) Poisson distribution



(b) Power-law distribution (log-log plot)

Figure 1.3: Two typical distributions

1.5 Models of complex networks

1.5.1 Regular Networks

The name of regular network comes from the mathematical definition of the regular graph where all nodes have the same degree. The typical regular network is a fully-connected network. This type of connection is used in computer networks to provide alternative path if the backbone line goes down. Properties of a fully-connected network [7]:

The average path length

$$L_full = 1, \tag{1.8}$$

the average clustering coefficient

$$C_full = 1, \tag{1.9}$$

Total number of edges of fully-connected network with N nodes is $N(N-1)/2$.

The other example of regular network is large-scale ring-shaped network where each node is connected to $2K$ nearest neighbours, where $K > 0$ but is still much smaller compared to N (number of vertices). Then it is possible to count its properties using the following equations.

The average path length (M is number of edges)

$$L_{ring} \approx \frac{M(M+1) - 2(K-1)(M-K+1)}{2M} \to \infty \qquad (M \to \infty) \tag{1.10}$$

Clustering coefficient of such a network

$$C_{ring} \approx \frac{3(K-1)}{2(2K-1)} \to \frac{3}{4} \qquad (K \to \infty) \tag{1.11}$$

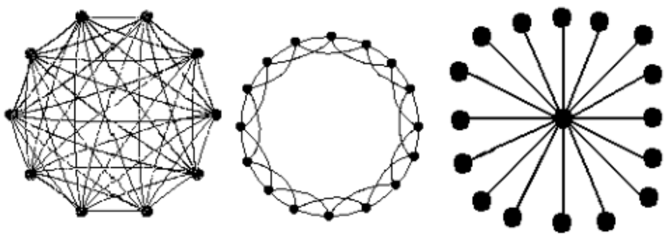


Figure 1.4: Some simple networks: (a) fully-connected network (b) ring-shaped network (c) star-shaped network

The last of most common regular network shape is called the *star-shaped* network (Fig. 1.4a). This topology is commonly used in computer networks, where the central node is switch or hub and the rest of nodes are clients stations or printers, etc...

For a star-shaped network of size N, counting the central node, the average path length is

$$L_{star} = 2 - \frac{2}{N} \to 2 \qquad (N \to 8) \tag{1.12}$$

and the clustering coefficient is

$$C_{star} = 0 \tag{1.13}$$

because all connections goes through one central node [7].

1.5.2 Random Networks

In 1959, P. Erdös and A. Rényi [1] introduced two random graph models. These models are $G(n, M)$ and $G(n, p)$. First model is based on given number of vertices and edges. Then the probability of the connection between two nodes with an edge is picked from the uniform space of all possible graphs containing n nodes and M edges. For example, let's have $G(3, 2)$ then the all possible graphs are $\binom{\binom{3}{2}}{2}$ and probability of creation one graph is $1/\binom{3}{2} = 1/3$.

To obtain a graph from the second model of ER random graph of size N. You need to pick up all possible pairs of nodes, once and once only, from the pool of N given nodes, and then connect each pair of nodes by an edge with probability p. Statistically, namely, after averaging over a large enough number of trials, this will yield a graph of N nodes and $pN(N-1)$ edges. Generally, the larger the p is, the denser the resultant network will be, as illustrated by Fig. 1.5. The figures shows that the different value of probability p

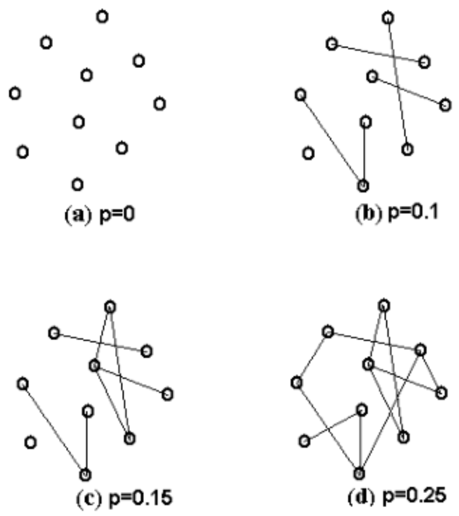


Figure 1.5: Some random graphs generated with different values of p [1]

gives different number of edges in graph. If the p is larger than the threshold $p_c \sim \ln N/N$ the graph will become connected and no separated nodes will left. ER random-graph networks have the following structural properties.

1. In an ER random-graph network, the average degree of nodes is

$$\langle k \rangle_{ER} = p(N-1) \tag{1.14}$$

2. the average path length is

$$L_{ER} \sim \frac{\ln N}{\ln \langle k \rangle_{ER}} \tag{1.15}$$

3. and the clustering coefficient is

$$C_{ER} \approx \frac{\langle k \rangle_{ER}}{N} = p \tag{1.16}$$

4. the node-degree distribution follows a Poisson distribution

$$P(k) = \frac{\mu^k}{k!} e^{-\mu} \tag{1.17}$$

in which μ is the expectation value $\mu = pN \approx \langle k \rangle_{ER}$.

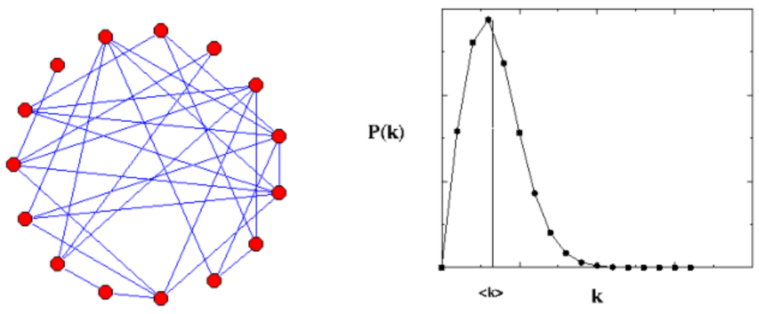


Figure 1.6: ER random graph and its degree distribution

1.5.3 Small-World Network Models

The term of small-world network come from many real-world experiments [8]. Imagine the situation, you are in a bus from a big city to the small town, the journey takes a long time and you eventually start talking to the unknown person next to you. Then after a few minutes of conversation you find out that the unknown person is good friend of your friend. "What a small world!"

This kind of experiment was made by Stanley Milgrams [9], psychologist of Harvard University. He randomly targeted two people unknown to each other, first one from a big city and the second one from small town. Then he selected another two groups of volunteers from different cities then two first candidates. And told them to send a letter to a friend and ask the friend to do the same with another friend. After few iterations of this game the letter reached the targets. Milgram found out that it took only 6 times in average to deliver the letter to each target. This experiment is well know as "six degrees of separation".

In 2011 another experiment of friendship investigation was made on 721 million active Facebook users. And it turns out that the separation between two random people is only 4.74 [10].

The small-world networks are not purely random but even not purely regular. So there was a need to set up a new model for the networks of this kind. To give ability to the scientist to simulate real world problems on computers.

1.5.3.1 The WS Small-World Network Model

From the examination of real-world networks came out that real networks have high clustering coefficient with short average path length. But the problem was the previous models of random and regular networks had only one property. Random networks appeared to have short average path length while regular lattices have high clustering.

The idea how to achieve desired network with high clustering and short average path properties was published by Duncan J. Watts and Steven H. Strogatz in the Nature magazine 1998 (Fig. 1.7b) [11].

They started transforming of regular ring-shaped network by tuning the value of connection probability p from 0 to 1. Where $p = 0$ refers to regular network and $p = 1$ is completely random network. The algorithm starts from ring-shaped network with N nodes in which each node is connected to its $2K$ neighbours. One end of each edge is or is not disconnected with a probability p and then reconnected to randomly chosen node, with avoidance of multi edges and self-loops. All nodes except the pair have same probability to become new neighbour of one node from selected pair.

1.5.3.2 The NW Small-World Network Mode

It was discovered the WS model can create a separate sub-networks according to according to edges disconnection. So one year later, M. Newman and D. Watts [12] modified the model by replacing disconnection and reconnection (rewiring) with addition of new edges (Fig. 1.7b). Then the $p = 0$ is original ring-shaped network and $p = 1$ yields a fully connected network.

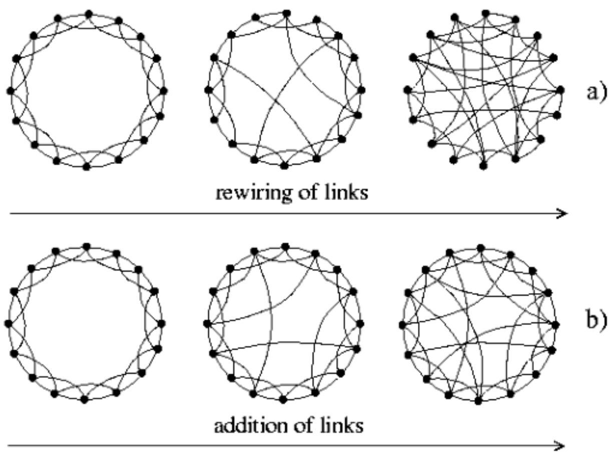


Figure 1.7: Illustration of two small-world network models:
(a) WS small-world network model (b) NW small-world network model

1.5.3.3 Statistical properties of small-world network

Clustering coefficient of WS model [6]

$$C(p) = \frac{3(K-1)}{2(2K-1)}(1-p^3) \tag{1.18}$$

clustering coefficient of the NW small-world network model

$$C(p) = \frac{3(K-2)}{4(K-1)+4Kp(p+2)}(1-p^3) \tag{1.19}$$

the average path length of the WS small-world network is

$$L(p) = \frac{2N}{K} f(\frac{2Np}{K}) \tag{1.20}$$

with

$$f(n) = \begin{cases} c & x \ll 1 \\ (3n+1)/2 & x \qquad (typically, c = 1/4) \\ gg1 & \end{cases} \tag{1.21}$$

the average path length of the NW small-world network model is also given by 1.20 where *f* is

$$f(x) \approx \frac{1}{2\sqrt{x^2+2x}} tanh^-1 \sqrt{\frac{x}{x^2+2x}} \tag{1.22}$$

degree distribution of the WS small-world network model

$$P_p(k) = \sum_{i=0}^{min(k-K,K)} \binom{K}{i} (1-p)^i \, p^{K-i} \frac{(Kp)^{k-K-i}}{(k-K-i)!} \exp(-Kp) \quad (k \geq K) \tag{1.23}$$

with $P_p(k) = 0$ for $k < K$

and the node-degree distribution of the NW small-world network

$$P(k) = \binom{N}{k-K}\left(\frac{Kp}{n}\right)^{k-K}\left(1-\frac{Kp}{N}\right)^{n-k+K} \quad (k \geq K) \tag{1.24}$$

with $P(k) = 0$ for $k < K$.

1.5.4 Scale-Free Network Models

All network models mentioned above do not grow in size, and their degree distribution follows closely to the Poisson distribution [8]. Poisson distribution, peaks at the average degree $\langle k \rangle$ and then decays exponentially fast on both sides as $k \to 0, \infty$. In particular, nodes with very high degrees $k \gg \langle k \rangle$ almost do not exist. Therefore these networks are called *homogeneous networks*. However, real networks do grow and some of them at a really fast rate [6].

A good example could be an Internet or cellular network. Both of these large scale networks have a lot of end users or devices (stubs, nodes with small degree), and much smaller number of hubs (transceivers, routers, nodes with high degree). The connectivity of growing scale-free networks is heterogeneous. That implies that the number of end nodes is growing much faster then the number of hubs. Also their degree distribution follows power-law formulation. The probability of breaking the network by the disconnection of random node is much lower, because most of nodes are end points, even the disconnection of a single hub may not to lead to the collapse of the entire network into separate elements.

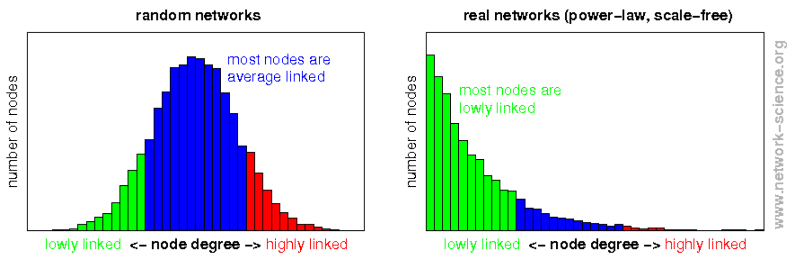
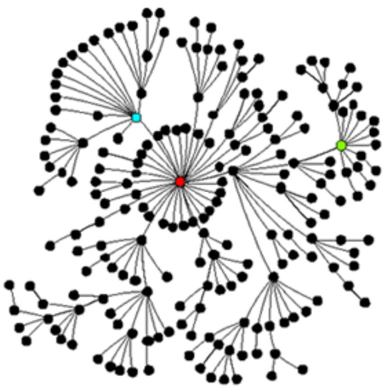


Figure 1.8: Poisson distribution and power-law degree distribution

From the figure 1.9a, the evolution of the network can be observed. The red central node is the main hub, where all other nodes started co-connecting to it and once the red node maximal capacity was reached, few of connected nodes became new hubs. As the network grew, new nodes tended to connect to the nodes with higher degree. This preferential connecting is called "rich gets richer" phenomenon. Assortative mixing implies that hub nodes have a greater probability of forming connections with other hub nodes 1.9b [6].


(a) scale-free network

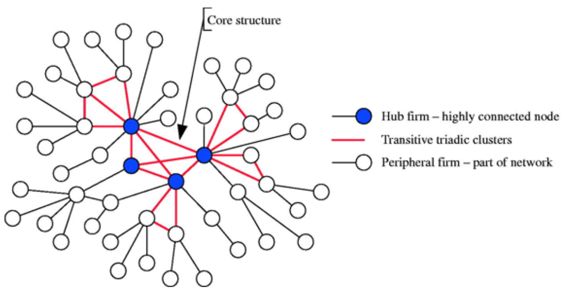

Core structure
Hub firm – highly connected node
Transitive triadic clusters
Peripheral firm – part of network
(b) A scale-free topology demonstrating a core-periphery structure

Figure 1.9: Two examples of scale-free networks

Based on this observation, the BA scale-free network were introduced.

1. *Growth*: Start from a fully-connected network of small size $m_0 \geq 1$, which has $\frac{m_0(m_0-1)}{2}$ edges; introduce one new node to the existing network each time, and this new node is connected to m existing nodes in the network simultaneously, where $1 \leq m \leq m_0$

2. *(Linear) Preferential Attachment*: The above incoming new node is connected to each of the m existing nodes, node i of degree k_i, according to the following probability:

$$\Pi_i = \frac{k_i}{\sum_{j=1}^{N} k_j} \tag{1.25}$$

Clearly, at the tth step of the node-adding process, the total number of existing nodes where $N = m_0 + t, t = 0, 1, 2, \ldots$

Here, the preferential attachment probability (3-20) is proportional to $k_i + 1$, so that even isolated node (with $k_i = 0$) has a non-zero probability to acquire new edges; otherwise it will be an isolated node forever. Of course, if the initial network is connected, then this $+1$ in (3-20) is no needed; nevertheless, comparing to a large number of N , this $+1$ does not change much in the formula anyway. Clearly, after t steps, the network will have $N = t + m_0$ nodes and $m \quad \frac{t+m_0(m_0-1)}{2}$ edges. Figure 3-9 illustrates the evolving process of a BA scale-free network, with $m = m_0 = 2$, where the black nodes represent the new comers and the white nodes are the existing ones at each step [6].

The average path length of the BA scale-free network is given as in Equation 1.26

$$L \frac{\ln N}{\ln \ln N} \tag{1.26}$$

The clustering coefficient of the BA scale-free network model is given by Equation 1.27.

$$C = \frac{m^2(m+1)^2}{4(m-1)} \left(\ln \left(\frac{m+1}{m} \right) - \frac{1}{m+1} \right) \frac{(\ln t)^2}{t} \tag{1.27}$$

The node-degree distribution of the BA scale-free network is approximately given by a power-law of the form in Equation 1.28.

$$P(k)\ 2m^2k^{-3} \tag{1.28}$$

1.6 Evolutionary Algorithms and Complex Networks

Exploration of the connection between complex network dynamics and evolutionary algorithms has been instigated by Zelinka et. al. [13]. This article first proposed the linkage between generation of complex network dynamics and possibility of visualisation of such dynamics using appropriate toolkits.

This work was further refined in [14], where the application of CML control was first introduced, and the possibility of control of evolutionary algorithms. The main imputes of the research has been the complex behaviour of the population, and whether it was possible to analyse such behaviour and obtain meaningful information, which could be then used to improve the algorithm.

The visualisation of the network was further described in [15], where the different attributes were compared and general conclusion asserting to the formulation of the complex network were described. This article reinforced the formulation of complex network dynamics inside EA's. An extended formulation encompassing all the basic test functions of real domain problem was given in [16].

These articles established a few preliminary facts:

1. that complex networks are formed in evolutionary algorithms
2. it is possible to visualise the network using complex network tools
3. relevant information can be obtained from such graphs
4. the graph generation is dependent on the algorithm and the problem

However, some of the issues that were not resolved, and therefore open for research was the problem type and application domain. All previous work has utilised the *continuous* variants of the algorithm. DE of [2] and SOMA of [4] have been used extensively. Additionally, only the standard data models of real domain systems was utilised [17].

The question then arose as to what effect does applying the *discrete* variants of these algorithms to *NP - Hard* problems have on the complex network generation. Is the complex network generated? What distinction is seen between the different EA's? Can any meaningful conclusion be ascertained from such graphs? This research is aimed at answering these questions.

2 Evolutionary Algorithms

2.1 Introduction

The genesis of EA's lies in the advent of modern day computers. A wide spectrum of such EA's exist and a basic outline is given in Fig. 2.1. This research introduces three different algorithms, each based on a novel paradigm.

The two algorithms of interest are the mathematical vector based DE and the predator-prey swarm based DSOMA. Both of these algorithms are introduced in the following sections.

Since a number of different algorithms are being introduced, an effort is made to used their own defined nomenclature. However, a schematic is presented in Fig. 2.2, where the general terminology is explained. The entire population is generally expressed as *P*, a candidate solution in the population is refereed to as an *individual* in the population, and a parameter in the individual is refereed to as an *element* in the individual. These terminology holds true, unless otherwise stated in the specific algorithms. The indexing uses the *row, column* notation; $x_{i,j}$, where i^{th} *row* represents the individual and the j^{th} *column* represents the specific element in the individual.

2.2 Differential Evolution

The DE algorithm introduced by Price *et. al.* [2] is a novel parallel direct search method, which utilises *NP* parameter vectors as a population for each generation *G*. DE can be categorised into a class of *floating-point encoded, evolutionary optimisation algorithms.* This section containing the description of DE and EDE contains extracts from [18].

Generally, the function to be optimised, $\Im$, is of the form $\Im(X) : R^D \to R$. The optimisation target is to minimise the value of this objective function $\Im(X)$,

$$\min\left(\Im(X)\right) \tag{2.1}$$

by optimising the values of its parameters $X = \{x_1, x_2, \ldots, x_D\}\,,\ X \in R^D$, where X denotes a vector composed of *D* objective function parameters. Usually, the parameters of the objective function are also subject to lower and upper boundary constraints, $x^{(L)}$ and $x^{(U)}$, respectively:

$$x_j{}^{(L)} \leq x_j \leq x_j{}^{(U)} \quad \forall j \in [1, D] \tag{2.2}$$

2.2.1 Initialisation

DE works with a population of candidate individuals, not with a single individual *solution* for the optimisation problem. Population *P* of generation *G* contains *NP* individual vectors called *individuals* of the population and each vector represents a potential solution for the optimisation problem:

$$P^{(G)} = X_i^{(G)} = x_{i,j}^{(G)} \quad j = 1, \ldots, D; \quad i = 1, \ldots, NP; \quad G = 1, \ldots, G_{\max} \tag{2.3}$$

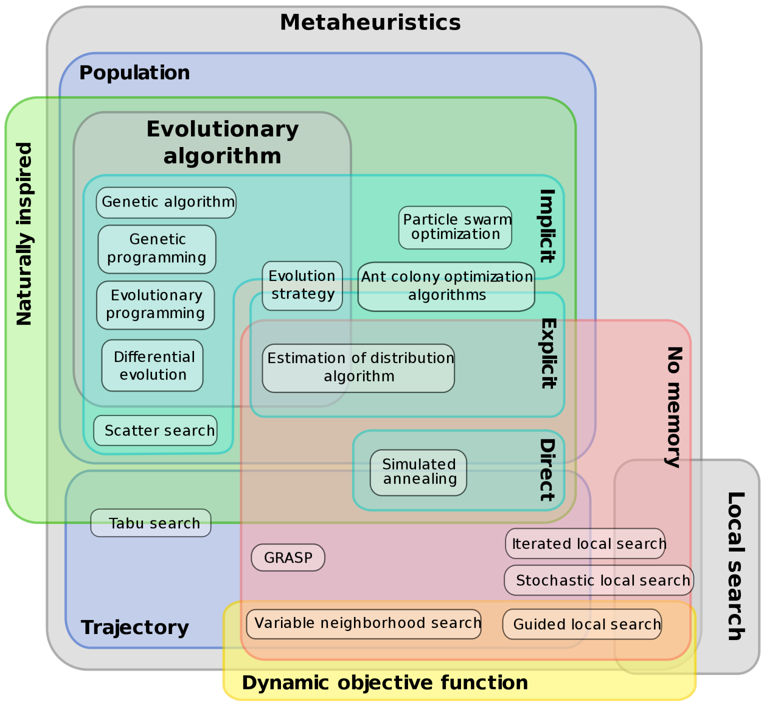


Figure 2.1: Metaheuristics Outline.

The natural way to initialise the population $P^{(0)}$ (initial population) is to seed it with random values within the given boundary constraints:

$$P^{(0)} = x_{i,j}^{(0)} = x_j^{(L)} + rand_j\,[0,1] \cdot \left(x_j^{(U)} - x_j^{(L)}\right) \quad \forall j \in [1, D]\,;\ \forall i \in [1, NP] \tag{2.4}$$

where $rand_j\,[0,1]$ represents a uniformly distributed random value that ranges from zero to one.

2.2.2 Mutation

The self-referential population recombination scheme of DE is different from the other EA's. From the first generation onward, the population of the subsequent generation $P^{(G+1)}$ is obtained on the basis of the current population $P^{(G)}$. First, a temporary or trial population of candidate vectors for the subsequent generation, $P'^{(G+1)} = V^{(G+1)} = v_{i,j}^{(G+1)}$, is generated as follows:

$$v_{i,j}^{(G+1)} = \begin{cases} x_{r_3,j}^{(G)} + F \cdot \left(x_{r_1,j}^{(G)} - x_{r_2,j}^{(G)}\right) if \ \ \mathrm{rand}_j\,[0,1] < CR \vee j = k \\ x_{i,j}^{(G)} \quad otherwise \end{cases} \tag{2.5}$$

where $j \in [1, D]$; $i \in [1, NP]$, $r_1, r_2, r_3 \in [1, NP]$ which are pairwise different from each other and current running index i, $k = (\mathrm{int}\,(rand_j\,[0.1] \cdot D) + 1)$, $CR \in [0, 1]$ and $F \in (0, 1]$.

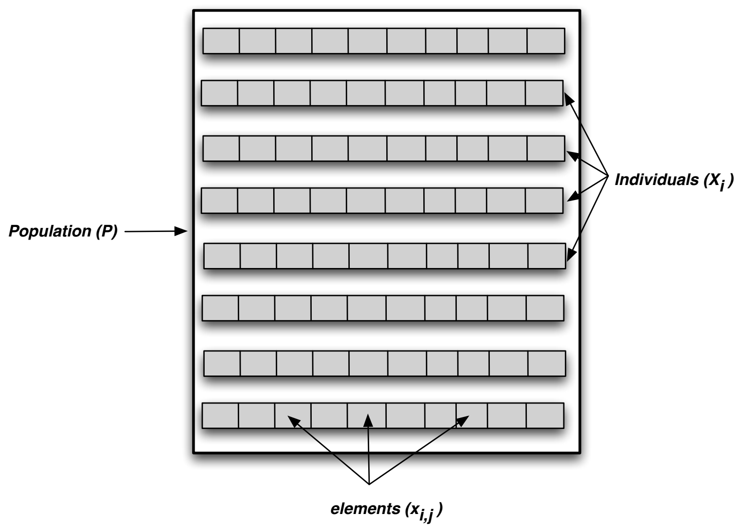


Figure 2.2: Generic population representation.

Three randomly chosen indexes, r_1, r_2, and r_3 refer to three randomly chosen vectors of population. They are pairwise different from each other and also different from the running index *i*. New random values for r_1, r_2, and r_3 are assigned for each value of index *i* (for each vector). A new value for the random number rand[0,1] is assigned for each value of index *j* (for each vector parameter).

2.2.3 Crossover

The index *k* refers to a randomly chosen vector parameter and it is used to ensure that at least one vector parameter of each individual trial vector $V^{(G+1)}$ differs from its counterpart in the previous generation $X^{(G)}$. A new random integer value is assigned to *k* for each value of the index *i* (prior to construction of each trial vector). *F* and *CR* are DE control parameters. Both values remain constant during the search process. Both values as well as the third control parameter, *NP* (population size), remain constant during the search process. *F* is a real-valued factor in range [0.0, 1.0] that controls the amplification of differential variations. *CR* is a real-valued crossover factor in the range [0.0, 1.0] that controls the probability that a trial vector will be selected from the randomly chosen, mutated vector, $v_{i,j}^{(G+1)}$ instead of from the current vector, $x_{i,j}^{(G)}$. Generally, both *F* and *CR* affect the convergence rate and robustness of the search process. Their optimal values are dependent both on objective function characteristics and on the population size, *NP*. Usually, suitable values for *F*, *CR* and *NP* can be found by experimentation after a few tests using different values.

2.2.4 Selection

The selection scheme of DE also differs from the other evolutionary algorithms. On the basis of the current population $P^{(G)}$ and the temporary population $P'^{(G+1)}$, the population of the next generation $P^{(G+1)}$ is created as follows:

$$X_i^{(G+1)} = \begin{cases} V_i^{(G+1)} if \ \Im\left(V_i^{(G+1)}\right) \leq \Im\left(X_i^{(G)}\right) \\ X_i^{(G)} \quad otherwise \end{cases} \tag{2.6}$$

Thus, each individual of the temporary or trial population is compared with its counterpart in the current population. The one with the lower value of cost function $\Im(X)$ to be minimised will propagate the population of the next generation. As a result, all the individuals of the next generation are as good or better than their counterparts in the current generation. The interesting point concerning the DE selection scheme is that a trial vector is only compared to one individual vector, not to all the individual vectors in the current population.

2.2.5 Boundary Constraints

It is important to notice that the recombination operation of DE is able to extend the search outside of the initialised range of the search space (equations (2.4) and (2.5)). It is also worthwhile to notice that sometimes this is a beneficial property in problems with no boundary constraints, because it is possible to find the optimum that is located outside of the initialised range. However, in boundary-constrained problems, it is essential to ensure that parameter values lie inside their allowed ranges after recombination. A simple way to guarantee this is to replace parameter values that violate boundary constraints with random values generated within the feasible range:

$$u_{i,j}^{(G+1)} = \begin{cases} x_j^{(L)} + rand_j\,[0,1] \cdot \left(x_j^{(U)} - x_j^{(L)}\right) \\ \qquad if \ \ u_{i,j}^{(G+1)} < x_j^{(L)} \vee u_{i,j}^{(G+1)} > x_j^{(U)} \\ u_{i,j}^{(G+1)} \ \ \text{otherwise} \end{cases} \tag{2.7}$$

2.2.6 DE Variants

Price [2] has suggested ten different working strategies. It mainly depends on the problem on hand for which strategy to choose. The strategies vary on the individual vectors to be perturbed, number of difference individual vectors under considered for perturbation, and finally the type of crossover used. The different strategies are outlined in Table 2.1.

The convention shown is DE/x/y/z. DE stands for Differential Evolution, *x* represents a string denoting the individual to be perturbed, *y* is the number of difference individuals considered for perturbation of *x*, and *z* is the type of crossover being used (exp: exponential; bin: binomial).

Table 2.1: DE Strategies

Strategy	Description
$DE/best/1/exp$	$v_i^{(G+1)} = x_{i_{best}}^{(G)} + F \cdot \left(x_{r_1}^{(G)} - x_{r_2}^{(G)}\right)$
$DE/rand/1/exp$	$v_i^{(G+1)} = x_{r_1}^{(G)} + F \cdot \left(x_{r_2}^{(G)} - x_{r_3}^{(G)}\right)$
$DE/rand-best/1/exp$	$v_i^{(G+1)} = x_i^{(G)} + \lambda \cdot \left(x_{i_{best}}^{(G)} - x_{r_1}^{(G)}\right) +$
	$F \cdot \left(x_{r_1}^{(G)} - x_{r_2}^{(G)}\right)$
$DE/best/2/exp$	$v_i^{(G+1)} = x_{i_{best}}^{(G)} + F \cdot \left(x_{r_1}^{(G)} - x_{r_2}^{(G)} - x_{r_3}^{(G)} - x_{r_4}^{(G)}\right)$
$DE/rand/2/exp$	$v_i^{(G+1)} = x_{r_5}^{(G)} + F \cdot \left(x_{r_1}^{(G)} - x_{r_2}^{(G)} - x_{r_3}^{(G)} - x_{r_4}^{(G)}\right)$
$DE/best/1/bin$	$v_i^{(G+1)} = x_{i_{best}}^{(G)} + F \cdot \left(x_{r_1}^{(G)} - x_{r_2}^{(G)}\right)$
$DE/rand/1/bin$	$v_i^{(G+1)} = x_{r_1}^{(G)} + F \cdot \left(x_{r_2}^{(G)} - x_{r_3}^{(G)}\right)$
$DE/rand-best/1/bin$	$v_i^{(G+1)} = x_i^{(G)} + \lambda \cdot \left(x_{i_{best}}^{(G)} - x_{r_1}^{(G)}\right) +$
	$F \cdot \left(x_{r_1}^{(G)} - x_{r_2}^{(G)}\right)$
$DE/best/2/bin$	$v_i^{(G+1)} = x_{i_{best}}^{(G)} + F \cdot \left(x_{r_1}^{(G)} - x_{r_2}^{(G)} - x_{r_3}^{(G)} - x_{r_4}^{(G)}\right)$
$DE/rand/2/bin$	$v_i^{(G+1)} = x_{r_5}^{(G)} + F \cdot \left(x_{r_1}^{(G)} - x_{r_2}^{(G)} - x_{r_3}^{(G)} - x_{r_4}^{(G)}\right)$

DE has two main phases of crossover: binomial and exponential. The *binomial* scheme takes parameters from the donor individual every time that the generated random number is less than the CR as given by $rand_j\,[0,1] < CR$, else all parameters come from the parent solution $x_i^{(G)}$. The *exponential* scheme takes the child individual vectors from $x_i^{(G)}$ until the first time that the random number is greater than *CR*, as given by $rand_j\,[0,1] < CR$, otherwise the parameters comes from the parent individual vector $x_i^{(G)}$. To ensure that each child individual vector differs from the parent individual vector, both the *exponential* and *binomial* schemes take at least one value from the mutated donor individual $v_i^{(G+1)}$.

The generic outline of DE is given in Fig. 2.3.

2.3 Enhanced Differential Evolution

Solving most combinatorial optimisation problems (of a permutative nature) requires discrete variables and an ordered sequence. To achieve this, two strategies known as forward and backward transformation techniques respectively were developed. The forward transformation method is described for transforming integer variables into continuous variables for the internal representation of vector values, since in its canonical form, the DE algorithm is only capable of handling continuous variables. The backward transformation method for transforming a population of continuous variables obtained after mutation back into integer variables for evaluating the objective function [19] is also described. Both forward and backward transformations are utilised in implementing the

```
1.Input :$D, G_{\max}, NP \geq 4, F \in (0, 1+), CR \in [0,1]$, and initial bounds :$x^{(L)}, x^{(U)}$.
2.Initialise : $\begin{cases} \forall i \leq NP \wedge \forall j \leq D : x_{i,j}^{(G=0)} = x_j^{(lo)} + rand_j\,[0,1] \cdot \left(x_j^{(U)} - x_j^{(L)}\right) \\ i = 1, 2, \ldots, NP,\ j = 1, 2, \ldots, D,\ G = 0, rand_j[0,1] \in [0,1] \end{cases}$
3.While $G < G_{\max}$
    $\forall i \leq NP$
        4. Mutate and recombine :
        4.1 $r_1, r_2, r_3 \in \{1, 2, \ldots, NP\}$,
            randomly selected and pairwise different
        4.2 $j_{rand} \in \{1, \ldots, D\}$, randomly selected once each $i$
        4.3 $\forall j \leq D, v_{i,j}^{(G+1)} = \begin{cases} x_{r_3,j}^{(G)} + F \cdot (x_{r_1,j}^{(G)} - x_{r_2,j}^{(G)}) \\ \quad \text{if} \quad (rand_j[0,1] < CR \vee j = j_{rand}) \\ x_{i,j}^{(G)} \quad \text{otherwise} \end{cases}$
        5. Select
            $X_i^{(G+1)} = \begin{cases} V_i^{(G+1)} & \text{if} \quad \Im(V_i^{(G+1)}) \leq \Im(X_i^{(G)}) \\ X_i^{(G)} & \text{otherwise} \end{cases}$
    $G = G + 1$
```

Figure 2.3: Canonical Differential Evolution Algorithm

EDE [20, 21, 22], algorithm. Fig. 2.4 shows how to deal with this inherent representational problem in DE. Level 0 deals with integer numbers (which are used in discrete problems). At this level, initialisation and final individuals are catered for. Level 1 of Fig. 2.4 deals with floating point numbers, which are suited for DE. At this level, the DE operators (mutation, crossover, and selection) take place. To transform the integer at level 0 into floating point numbers at level 1 for DE's operators, requires some specific kind of coding. This type of coding is highly used in mathematics and computing science. For the basics of transforming an integer number into its real number equivalence, interested readers may refer to [23], and [24] for its application to optimising machining operations using genetic algorithms.

The general outline is given below

1. **Initial Phase**

    (a) *Population Generation*: An initial number of discrete parent individuals are generated for the initial population.

2. **Conversion**

    (a) *Forward transformation*: This conversion schema transforms the parent individual into the required continuous parent individual.

    (b) *DE Strategy*: The DE strategy transforms the continuous parent individual into the continuous child individual using its inbuilt crossover and mutation schemas.

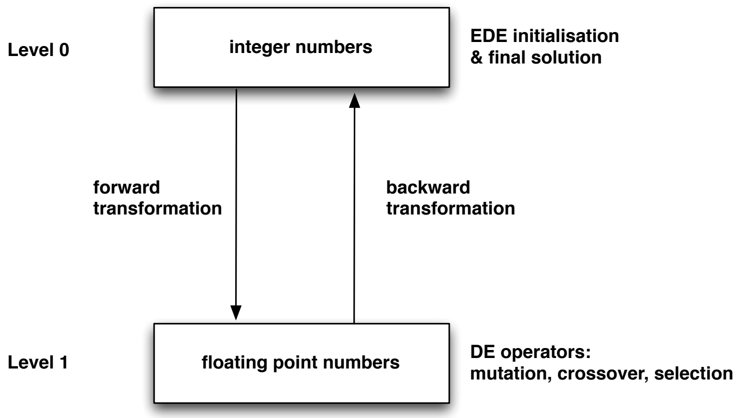


Figure 2.4: Metaheuristics Outline.

   (c) *Backward transformation*: This conversion schema transforms the continuous child individual into a discrete child individual.

3. **Repairment**

   (a) *Random repairment*: Repairs the child individual into the discrete individual of unique values.

4. **Improvement Strategy**

   (a) *Mutation*: Standard mutation is applied to obtain a better individual.
   (b) *Insertion*: Uses a two-point cascade to obtain a better individual.

5. **Local Search**

   (a) *Local Search*: 2 Opt local search is used to explore the neighbourhood of the individual.

2.3.1 Forward transformation

In integer variable optimisation a set of integer number is normally generated randomly as an initial individual. Let this set of integer number be represented as:

$$z'_i \in z' \tag{2.8}$$

Let the real number (floating point) equivalence of this integer number be z_i. The length of the real number depends on the required precision, which in this case, is chosen two places after the decimal point. The domain of the variable z_i has length equal to 5; the precision requirement implies that the range be [0,...,4]. Although 0 is considered since it is not a feasible individual, the range [0.1, 1, 2, 3, 4] is chosen, which gives a range of 5. For each feasible individual two decimal places are assigned and this gives 5 x 100 = 500 . Accordingly, the equivalent continuous variable for ${z'}_i$ is given as

$$100 = 10^2 < 5 \cdot 10^2 \leq 10^3 = 1000 \tag{2.9}$$

The mapping from an integer number to a real number z_i straightforward, for the given range is now given as

$$z_i = -1 + \frac{{z'}_i \cdot 5}{10^3 - 1} \tag{2.10}$$

Equation 2.10 results in most conversion values being negative; this does not create any accuracy problem any way. After some studies [19], the scaling factor f = 100 was found to be adequate for converting virtually all integer numbers into their equivalent positive real numbers. Applying this scaling factor of f = 100 gives

$$z_i = -1 + \frac{{z'}_i \cdot f \cdot 5}{10^3 - 1} \tag{2.11}$$

Equation (2.11) is used to transform any integer variable into an equivalent continuous variable, which is then used for the DE internal representation of the population of vectors. Without this transformation, it is not possible to make useful moves towards the global optimum in the solution space using the mutation mechanism of DE, which works better on continuous variables.

2.3.2 Backward Transformation

Integer variables are used to evaluate the objective function. The DE self-referential population mutation scheme is quite unique. After the mutation of each vector, the trial vector is evaluated for its objective function in order to decide whether or not to retain it. This means that the objective function values of the current vectors in the population need to be also evaluated. These vector variables are continuous (from the forward transformation scheme) and have to be transformed into their integer number equivalence. The backward transformation technique is used for converting floating point numbers to their integer number equivalence. The scheme is given as follows:

$${z'}_i = \frac{(1 + z_i) \cdot \left(10^3 - 1\right)}{500} \tag{2.12}$$

In this present form the backward transformation function is not able to properly discriminate between variables. To ensure that each number is discrete and unique, some modifications are required as follows:

$$\begin{aligned} &\alpha = \text{int}\,({z'}_i + 0.5) \\ &\beta = \alpha - {z'}_i \\ &z_i^* = \begin{cases} (\alpha - 1) & if\ \beta > 0.5 \\ \alpha & if\ \beta < 0.5 \end{cases} \end{aligned} \tag{2.13}$$

Equation (2.13) gives z_i^*, which is the transformed value used for computing the objective function. It should be mentioned that the conversion scheme of equation (2.12), which transforms real numbers after DE operations into integer numbers is not sufficient to avoid duplication; hence, the steps highlighted in equation (2.13) are important. In our studies, these modifications ensure that after mutation, crossover and selection operations, the converted floating numbers into their integer equivalence in the set of jobs for a new scheduling solution, or set of cities for a new TSP solution, etc., are not duplicated.

Only the main and significant aspects of the EDE; forward and backward transformation routines are presented here. For a detailed description, including pseudocode, worked examples and working codes, the interested reader is directed to [18] and [3].

2.3.3 2 Opt local search

EDE employs the 2 Opt local search as its improvement strategy. If the best individual does not improve over the previous five consecrative generations, the local search is applied to all the individuals in the subsequent iteration in order to push the population out of the local minima. The 2 Opt local search is a very simple routine. It utilises two iterators; one for the outer loop $k = 1, 2, \ldots, D - 1$ and one for the inner loop $l = k + 1, \ldots, D$. Using the iterators as indexes to the individuals, the indexed variables in the individuals are exchanged $X_i^{(G)} = \left\{\ldots, x_{i,l}^{(G)}, \ldots, x_{i,k}^{(G)}, \ldots\right\}$ and the new trial individual evaluated. If any improvement is seen, the individual is adapted as the main individual under consideration. The complexity of this local search is $O\left(n^2\right)$, where n is the input schedule.

2.4 Self-Organising Migrating Algorithm

SOMA [4] is a metaheuristic, which is based on the competitive-cooperative behaviour of intelligent creatures solving a common problem.

In SOMA, individuals reside in the optimised model's hyperspace, with the objective to find the best individual within this space. It can be said, that this kind of behaviour of intelligent individuals allows SOMA to realise very successful search.

Because SOMA uses the philosophy of competition and cooperation, the variants of SOMA are called strategies. They differ in the way as to how the individuals affect all others. Taking t is the iterator for the migrations M ($t = 1, \ldots, M$), i as the index to the individuals in the populations ($i = 1, \ldots, \beta$) and j as the index to each element in the individual $j = 1, \ldots, N$, SOMA can be described as consisting of the following steps:

1. *Definition of parameters*. Before execution, the SOMA parameters (PathLength, Step, PRT, Migrations see Table 2.2) are defined.

2. *Creating of population.* The population ($P = \left\{X_1^t, X_2^t, \ldots, X_\beta^t\right\}$) is generated consisting of a number of individuals (β), where each individual ($X_i^t = \left\{x_{i,1}^t, x_{i,2}^t, \ldots, x_{i,N}^t\right\}$), represented as $x_{i,j}^t : i = 1, \ldots, \beta; j = 1, \ldots, N$ and $t = 1, \ldots, M$, contains a number of elements (N), which is the dimension of the problem.

Table 2.2: SOMA parameters.

Name	Range	Type
PathLength	$(1.1 - 3)$	Control
StepSize	$(0.11 -$ PathLength$)$	Control
PRT	$(0 - 1)$	Control
MinDiv	Negative - User Defined	Termination
Migrations	10+	Termination

3. *Migration loop.*
   (a) Each individual X_i^t is evaluated for the cost function, using the function $\Im(X)$ and the fitness value stored in the cost fitness matrix C:
   $$C_i^t = \Im\left(X_i^t\right), \quad i = 1, \dots, \beta$$
   (b) For each element $x_{i,j}^t$ in an individual X_i^t, the perturbation matrix $\mathbf{A}_{i,j}(PRT)$ is created (2.14).
   (c) All individuals perform their run towards the selected individual (Leader), which has the best fitness for that migration according to (2.15). Each individual is selected piecewise. The movement consists of jumps determined by the step parameter (s) until the individual reaches the final position given by the PathLength parameter. For each step, the cost function for the actual position is evaluated and the best value is saved. Then, the individual returns to the position, where it found the best-cost value on its trajectory.

SOMA, like other EA's, is controlled by a number of parameters, which are predefined. They are presented in Table 2.2.

2.4.1 Mutation

Mutation, the random perturbation of individuals, is applied differently in SOMA compared with other evolutionary strategies. SOMA uses a parameter called PRT to achieve perturbation. It is defined in the range [0, 1] and is used to create a perturbation matrix ($\mathbf{A}_{i,j}(PRT)$) (2.14):

$$\mathbf{A}_{i,j} = \begin{cases} 1 & \text{if rand}() < PRT \\ 0 & \text{otherwise} \end{cases} \qquad (2.14)$$
$$j = 1, 2, \dots, N; \; i = 1, 2, \dots, \beta$$

The novelty of this approach is that in its canonical form, the perturbation matrix is created before an individual starts its journey over the search space. The perturbation matrix defines the final movement of an active individual in search space.

The randomly generated binary perturbation matrix controls the allowed dimensions for an individual. If an element of the perturbation matrix is set to zero, then the individual is not allowed to change its position in the corresponding dimension.

2.4.2 Crossover

In standard evolutionary strategies, the crossover operator usually creates new individuals based on information from the previous generation. Geometrically speaking, new positions are selected from an N-dimensional hyper-plane. In SOMA, which is based on the simulation of cooperative behaviour of intelligent beings, sequences of new positions in the N-dimensional hyperplane are generated. The movement of an element is thus given as follows:

$$x_{i,j}^{t} = x_{i,j}^{t-1} + \left(x_{L,j}^{t-1} - x_{i,j}^{t-1}\right) \cdot s \cdot \mathbf{A}_{i,j}, \quad j = 1,2,\ldots,N;\ i = 1,2,\ldots,\beta \tag{2.15}$$

where:

- $x_{i,j}^{t}$: new candidate individual
- $x_{i,j}^{t-1}$: original individual
- $x_{L,j}^{t-1}$: leader individual
- s : $\in$ [0 , Path length]
- $\mathbf{A}$: control perturbation matrix

It can be observed from equation (2.15) that the perturbation matrix causes an individual to move towards the leading individual (the one with the best fitness) in *N-k* dimensional space. If all N elements of the perturbation matrix are set to 1, then the search process is carried out in an N dimensional hyperplane (*i.e.* on a *N+1* fitness landscape). If some elements of the perturbation matrix are set to 0, then the second terms on the right−hand side of (2.15) equals 0. This implies that those parameters of an individual that are related to 0 in the perturbation matrix are not changed during the search. The number of frozen parameters, *k*, is simply the number of dimensions that are not taking part in the actual search process. Therefore, the search process takes place in a *N-k* dimensional subspace. Fig. 2.5 shows the directional jump sequences of the different individuals towards the leader. Fig. 2.6 shows the end position of the individuals upon the completion of the migrations. The overall schematic of the SOMA migration is given in Fig. 2.7.

2.5 Discrete Self-Organising Migrating Algorithm

DSOMA [25] is the discrete version of SOMA, developed to solve permutation based combinatorial optimisation problem. The same ideology of the sampling of the space between two individuals is retained. Assume that there are two individuals in a search space as given in Fig. 2.8. The objective for DSOMA is to transverse from one individual to another, while mapping each discrete space between these two individuals. Figs. 2.9 and 2.10 are 3D representations, where the DSOMA mapping is shown as the surface joining these two.

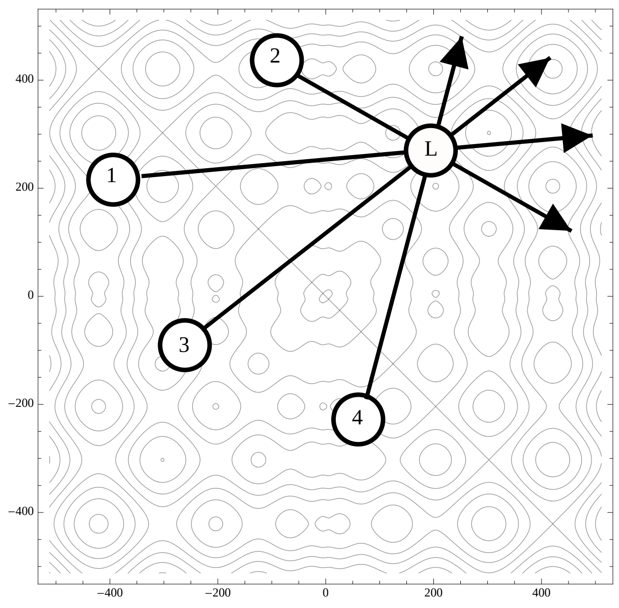


Figure 2.5: SOMA jump sequence.

The major input of this algorithm is the sampling of the jump sequence between the individuals in the populations, and the procedure of constructing new trial individuals from these sampled jump sequence elements.

The overall outline for DSOMA can be given as:

1. **Initial Phase**
   (a) *Population Generation*: An initial number of permutative trial individuals is generated for the initial population.
   (b) *Fitness Evaluation*: Each individual is evaluated for its fitness.
2. **DSOMA**
   (a) *Creating Jump Sequences*: Taking two individuals, a number of possible jump positions is calculated between each corresponding element.
   (b) *Constructing Trial Individuals*: Using the jump positions; a number of trial individuals is generated. Each element is selected from a jump element between the two individuals.
   (c) *Repairment*: The trial individuals are checked for feasibility and those, which contain an incomplete schedule, are repaired.
3. **Selection**
   (a) *New Individual Selection*: The new individuals are evaluated for their fitness and the best new fitness based individual replaces the old individual, if it improves upon its fitness.

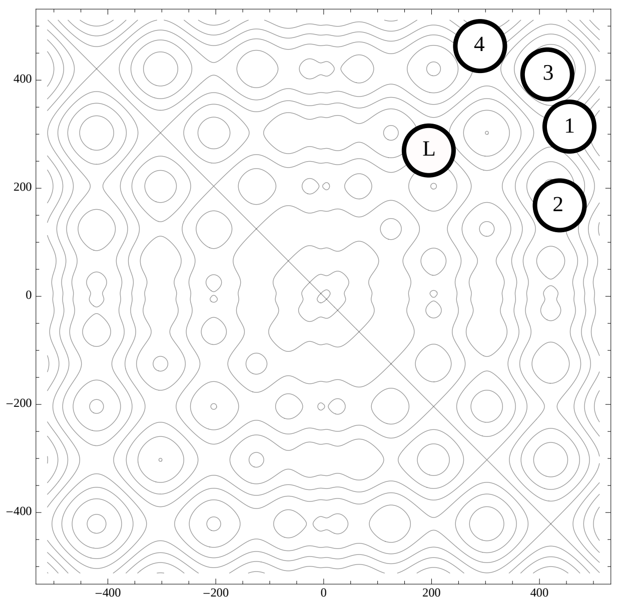


Figure 2.6: SOMA end jump sequence.

Table 2.3: DSOMA parameters.

Name	Range	Type	
J_{min}	(1+)	Control	Minimum number of jumps
Population	10+	Control	Number of individuals
Migrations	10+	Termination	Total number of iterations

4. **Generations**
   (a) *Iteration*: Iterate the population till a specified migration.

DSOMA requires a number of parameters as given in Table 2.3. The major addition is the parameter J_{min}, which gives the minimum number of jumps (sampling) between two individuals. The SOMA variables PathLength, StepSize and PRT Vector are not initialised as they are dynamically calculated by DSOMA using the adjacent elements between the individuals.

2.5.1 Initialisation

The population is initialised as a permutative schedule representative of the size of the problem at hand (2.16). As this is the initial population, the superscript of x its set to 0. The *rand()* function obtains a value between 0 and 1, and the *INT()* function rounds

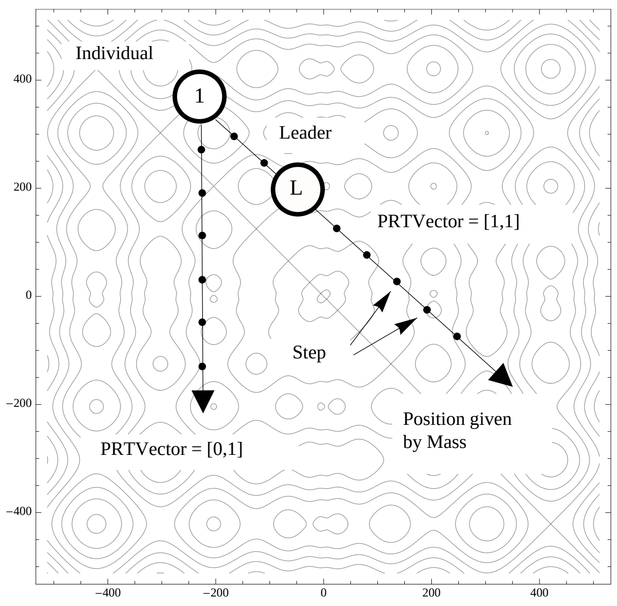


Figure 2.7: SOMA generic search sequence.

down the real number to the nearest integer. The *if* condition checks to ensure that each element within the individual is unique.

$$x_{i,j}^0 = \left\{ \begin{array}{l} 1 + INT\left(rand\left(\right) \cdot (N-1)\right) \\ \text{if } \; x_{i,j}^0 \notin \left\{ x_{i,1}^0, \ldots, x_{i,j-1}^0 \right\} \\ i = 1, \ldots, \beta; \; j = 1, \ldots, N \end{array} \right. \tag{2.16}$$

Each individual is vetted for its fitness (2.17), and the best individual, whose index in the population can be assigned as L (leader) and it is designated the leader as X_L^0 with its best fitness given as C_L^0.

$$C_i^0 = \Im\left(X_i^0\right), \quad i = 1, \ldots, \beta \tag{2.17}$$

The pseudocode for generating a population is given in Fig. 2.11.

After the generation of the initial population, the migration counter *t* is set to 1 where $t = 1, \ldots, M$ and the individual index *i* is initialised to 1, where $i = 1, \ldots, \beta$. Using these values, the following sections (section 2.5.2 - section 2.5.5) are recursively applied, with the counters *i* and *t* being updated in section 2.5.6 and section 2.5.7 respectively.

2.5.2 Creating Jump Sequences

DSOMA operates by calculating the number of discrete jump steps that each individual has to circumnavigate. In DSOMA, the parameter of minimum jumps ($J_{\min}$) is used in lieu of PathLength, which states the minimum number of individuals or sampling between the two individuals.

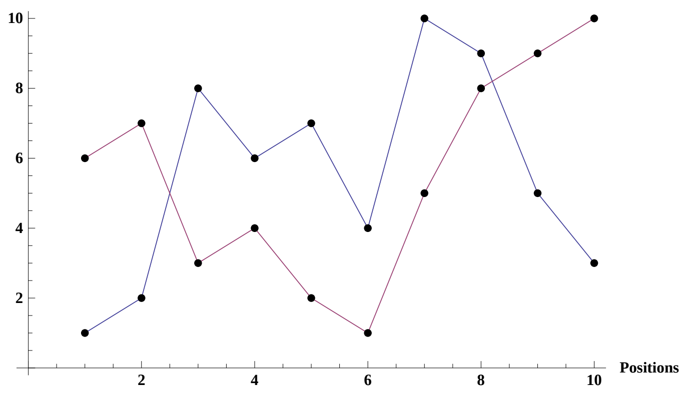


Figure 2.8: Two individuals in search space.

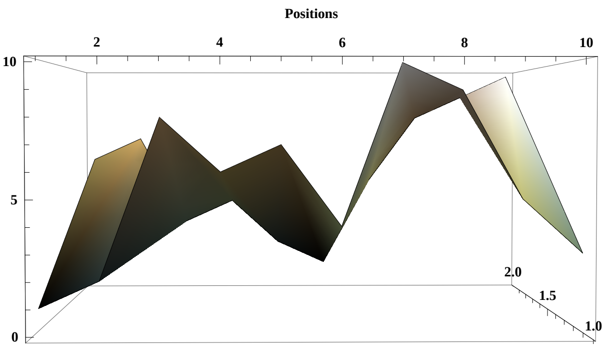


Figure 2.9: End view of the two individuals in 3D search space.

Taking two individuals in the population, one as the incumbent (X_i^t) and the other as the leader (X_L^t), the possible number of jump individuals $J_{\max}$ is the mode of the difference between the adjacent values of the elements in the individual (2.18). A vector *J* of size *N* is created to store the difference between the adjacent elements in the individuals. The *mode ()* function obtains the most common number in *J* and designates it as J_{max}.

$$\begin{aligned} J_j &= \left| x_{i,j}^{t-1} - x_{L,j}^{t-1} \right|, \quad j = 1, \ldots, N \\ J_{\max} &= \begin{cases} mode\,(J) \;\; if \;\; mode\,(J) > 0 \\ 1 \;\; otherwise \end{cases} \end{aligned} \tag{2.18}$$

The step size (*s*), can now be calculated as the integer fraction between the required jumps and possible jumps (2.19).

$$s = \begin{cases} \left\lfloor \frac{J_{\max}}{J_{\min}} \right\rfloor & \text{if } J_{\max} \geq J_{\min} \\ 1 & \text{otherwise} \end{cases} \tag{2.19}$$

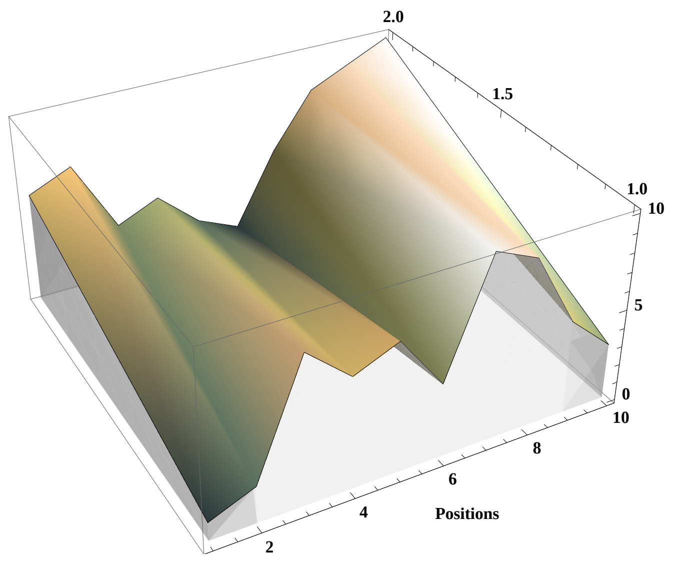


Figure 2.10: Isometric view of the two individuals in 3D search space.

Create a jump matrix $\mathbf{G}$, which contains all the possible jump positions, that can be calculated as:

$$\mathbf{G}_{l,j} = \begin{cases} x_{i,j}^{t-1} + s \cdot l & \text{if } x_{i,j}^{t-1} + s \cdot l < x_{L,j}^{t-1} \text{ and } x_{i,j}^{t-1} < x_{L,j}^{t-1} \\ x_{i,j}^{t-1} - s \cdot l & \text{if } x_{i,j}^{t-1} + s \cdot l < x_{L,j}^{t-1} \text{ and } x_{i,j}^{t-1} > x_{L,j}^{t-1} \\ 0 & \text{otherwise} \end{cases} \tag{2.20}$$
$$j = 1, \ldots, N; \; l = 1, \ldots, J_{\min}$$

The pseudocode for creating jump sequences is given in Fig. 2.12.

2.5.3 Constructing Trial Individuals

For each jump sequence of two individuals, a total of $J_{\min}$ new individuals can now be constructed from the jump positions. Taking a new temporary population H ($H = \{Y_1, \ldots, Y_{J_{\min}}\}$), in which each new individual Y_w ($w = 1, \ldots, J_{\min}$), is constructed piecewise from $\mathbf{G}$. Each element $y_{w,j}$ ($Y_w = \{y_{w,j}, \ldots, y_{w,N}\}$, $j = 1, 2, \ldots, N$) in the individual, indexes its values from the corresponding j^{th} column in $\mathbf{G}$. Each l^{th} ($l = 1, \ldots, J_{\min}$) position for a specific element is sequentially checked in $\mathbf{G}_{l,j}$ to ascertain if it already exists in the current individual Y_w. If this is a new element, it is then accepted in the individual, and the corresponding l^{th} value is set to zero as $\mathbf{G}_{l,j} = 0$. This iterative procedure can be given as in equation (2.21) and the pseudocode for constructing trial individual is represented in Fig. 2.13.

$$y_{w,j} = \begin{cases} \mathbf{G}_{l,j} \begin{cases} \text{if} \quad \mathbf{G}_{l,j} \notin \; \{y_{w,1}, \ldots, y_{w,j-1}\} \text{ and } \mathbf{G}_{l,j} \neq 0; \\ \text{then } \mathbf{G}_{l,j} = 0; \end{cases} \\ 0 \quad \text{otherwise} \end{cases} \tag{2.21}$$
$$l = 1, \ldots, J_{\min}; \; j = 1, \ldots, N; \; w = 1, \ldots, J_{\min}$$

Pseudocode for Generating Initial Population

Assume a population P of size β, a problem of size N, and a schedule given as $X = \{x_1, \ldots, x_N\}$. The fitness can be given as C, while the best individual is represented as X_L with its associated fitness C_L and its index b.

```
1. For $i = 1, 2, \ldots, \beta$ do the following:
    (a) For $j = 1, 2, \ldots, N$ do the following:
        i. Randomly generate a value $x_j = \mathrm{rnd\ int\ [1, N]}$
        ii. **WHILE** $x_j \notin X_i$
                Randomly generate a value $x_j = \mathrm{rnd\ int\ [1, N]}$
        iii. Insert $x_j \to X_i$
    (b) Insert $X_i \to P$
    (c) Calculate the fitness of $X_i$ as $C_i = \Im(X_i)$
2. Set $C_L$ = min$(C)$
3. Set best index $b$ = index of min$(C)$
4. Set $X_L$ = $X_b$
```

Figure 2.11: Pseudocode for Generating Initial Population.

2.5.4 Repairing Trial Individuals

Some individuals may exist, which may not contain a permutative schedule. The jump individuals Y_w $(w = 1, 2, \ldots, J_{min})$, are constructed in such a way, that each infeasible element $y_{w,j}$ is indexed by 0.

Taking each jump individual Y_w iteratively from H, the following set of procedures can be applied recursively.

Take A and B, where A is initialised to the permutative schedule $A = \{1, 2, \ldots, N\}$ and B is the complement of individual Y_w relative to A as given in equation (2.22).

$$B = A \backslash Y_w \tag{2.22}$$

If after the complement operation, B is an empty set without any elements; $B = \{\}$, then the individual is correct with a proper permutative schedule and does not require any repairment.

However, if B contains values, then these values are the missing elements in individual Y_w. The repairment procedure is now outlined. The first process is to randomise the positions of the elements in set B. Then, iterating through the elements $y_{w,j}$ $(j = 1, \ldots, N)$ in the individual Y_w, each position, where the element $y_{w,j} = 0$ is replaced by the value in

B. Assigning B_{size} as the total number of elements present in B (and hence missing from the individual Y_w), the repairment procedure can be given as in equation (2.23).

$$y_{w,j} = \begin{cases} B_h & \text{if } y_{w,j} = 0 \\ y_{w,j} & \text{otherwise} \end{cases} \qquad (2.23)$$
$$h = 1, \ldots, B_{size}; \; j = 1, \ldots, N$$

After each individual is repaired in H, it is then evaluated for its fitness value as in equation (2.24) and stored in γ, the fitness array of size J_{min}.

$$\gamma_w = \Im\left(Y_w\right), \quad w = 1, \ldots, J_{\min} \qquad (2.24)$$

The pseudocode for repairing trial individuals is given in Fig. 2.14.

2.5.5 Population update

2 Opt local search is applied to the best individual Y_{best} obtained with the minimum fitness value ($\min\left(\gamma_w\right)$). After the local search routine, the new individual is compared with the fitness of the incumbent individual X_i^{t-1}, and if it improves on the fitness, then the new individual is accepted in the population (2.25).

$$X_i^t = \begin{cases} Y_{best} & \text{if } \Im\left(Y_{best}\right) < C_i^{t-1} \\ X_i^{t-1} & \text{otherwise} \end{cases} \qquad (2.25)$$

If this individual improves on the overall best individual in the population, it then replaces the best individual in the population (2.26).

$$X_{best}^t = \begin{cases} Y_{best} & \text{if } \Im\left(Y_{best}\right) < C_{best}^t \\ X_{best}^{t-1} & \text{otherwise} \end{cases} \qquad (2.26)$$

2.5.6 Iteration

Sequentially, incrementing i, the population counter by 1, another individual X_{i+1}^{t-1} is selected from the population, and it begins its own sampling towards the designated leader X_L^{t-1} from sections 2.5.2 - 2.5.5. It should be noted that the leader does not change during the evaluation of one migration.

2.5.7 Migrations

Once all the individuals have executed their sampling towards the designated leader, the migration counter t is incremented by 1. The individual iterator i is reset to 1 (the beginning of the population) and the loop in sections 2.5.2 - 2.5.6 is re-initiated.

2.5.8 2 Opt local search

The local search utilised in DSOMA is the same 2 Opt local search algorithm used in EDE (section 2.3.3). The reason as to why the 2 Opt local search was chosen, is that it is the simplest in the *k*-opt class of routines. As the DSOMA sampling occurs between two individuals in *k*-dimension, the local search refines the individual. This in turn provides a better probability to find a new leader after each jump sequence. The placement of the local search was refined heuristically during experimentation.

The complexity of this local search is $O\left(n^2\right)$. As local search makes up the majority of the complexity time of DSOMA, the overall computational complexity of DSOMA for a single migration is $O\left(n^3\right)$.

A schematic of the DSOMA routine is given in Fig. 2.15, which graphically outlines the procedure for creating jump sequence between two individuals, and constructing trial individuals.

Pseudocode for Creating Jump Sequences

Take the population *P* with its associated fitness *C*. The number of jumps (PathLength) is given as J_{min} and the step size as s. Assume an empty schedule for storing the possible jump sequences as *J* and a temporary jump matrix to store the calculated individuals as *G*, whose size is J_{min} by *N*, were *N* is the size of the individual.

```
1. For $i = 1, 2, \ldots, \beta$ do the following:
   (a) For $j = 1, 2, \ldots, N$ do the following:
       i. $J_j = |x_{L,j} - x_{i,j}|$
   (b) $J_{max} =$ mode $(J)$
   (c) IF $J_{\max} \geq J_{\min}$
          $s = \left\lfloor \frac{J_{\min}}{J_{\max}} \right\rfloor$
       ELSE
          $s = 1$
   (d) For $j = 1, 2, \ldots, N$ do the following:
       i. For $l = 1, 2, \ldots, J_{min}$ do the following:
          A. IF $x_{i,j} < x_{L,j}$
                $\mathbf{G}_{l,j} = x_{i,j} + s \cdot l$
             ELSE IF $x_{i,j} > x_{L,j}$
                $\mathbf{G}_{l,j} = x_{i,j} - s \cdot l$
             ELSE
                $\mathbf{G}_{l,j} = 0$
       ii. Create New Trial Individual as given in Fig. 2.13.
```

Figure 2.12: Pseudocode for Creating Jump Sequences.

Pseudocode for Constructing Trial Individuals

Take the temporary jump matrix G, of size is J_{min} by N, which is now populated with calculated jump sequence elements. Create a temporary population $H = \{Y_1, \ldots, Y_{J_{\min}}\}$, where $Y = \{y_1, \ldots, y_N\}$

```
1. For $w = 1, 2, \ldots, J_{min}$ do the following:
   (a) For $j = 1, 2, \ldots, N$ do the following:
       i. For $l = 1, 2, \ldots, J_{min}$ do the following:
          A. IF ($\mathbf{G}_{l,j} \notin \{y_{w,1}, \ldots, y_{w,y-1}\}$ AND $\mathbf{G}_{l,j} \neq 0$)
                $y_{w,j} = \mathbf{G}_{l,j}$
                $\mathbf{G}_{l,j} = 0$
       ii. IF $y_{w,j} ==$ NULL
             $y_{w,j} = 0$
2. Repair the temporary population $Y$ in Fig. 2.14.
```

Figure 2.13: Algorithm for Constructing Trial Individuals.

Pseudocode for Repairing Trial Individuals

Take the temporary population H and its associated fitness array γ of size J_{min}. Assume two schedules A and B of maximum size N, where A is initialised to $A = \{1, 2, \ldots, N\}$. The best new trial individual is represented as Y_{best}, with its fitness $\Im\left(Y_{best}\right)$.

```
1. For $w = 1, 2, \ldots, J_{min}$ do the following:
    (a) $B = A \backslash Y_w$
    (b) **IF** $B = \{\}$
        Randomise the elements in $B$.
        i. For $j = 1, 2, \ldots, N$ do the following:
            A. SET index $h = 1$
            B. **IF** $y_{w,j} == 0$
                   $y_{w,j} = B_h$
                   $h = h + 1$
    (c) Evaluate the fitness of the new trial individual as:
            $\gamma_w = \Im(Y_w)$
    (d) **IF** $w == 1$
            $Y_{best} = Y_w$
    (e) **ELSE IF** $\gamma_w < Y_{best}$
            $Y_{best} = Y_w$
```

Figure 2.14: Pseudocode for Repairing Trial Individuals.

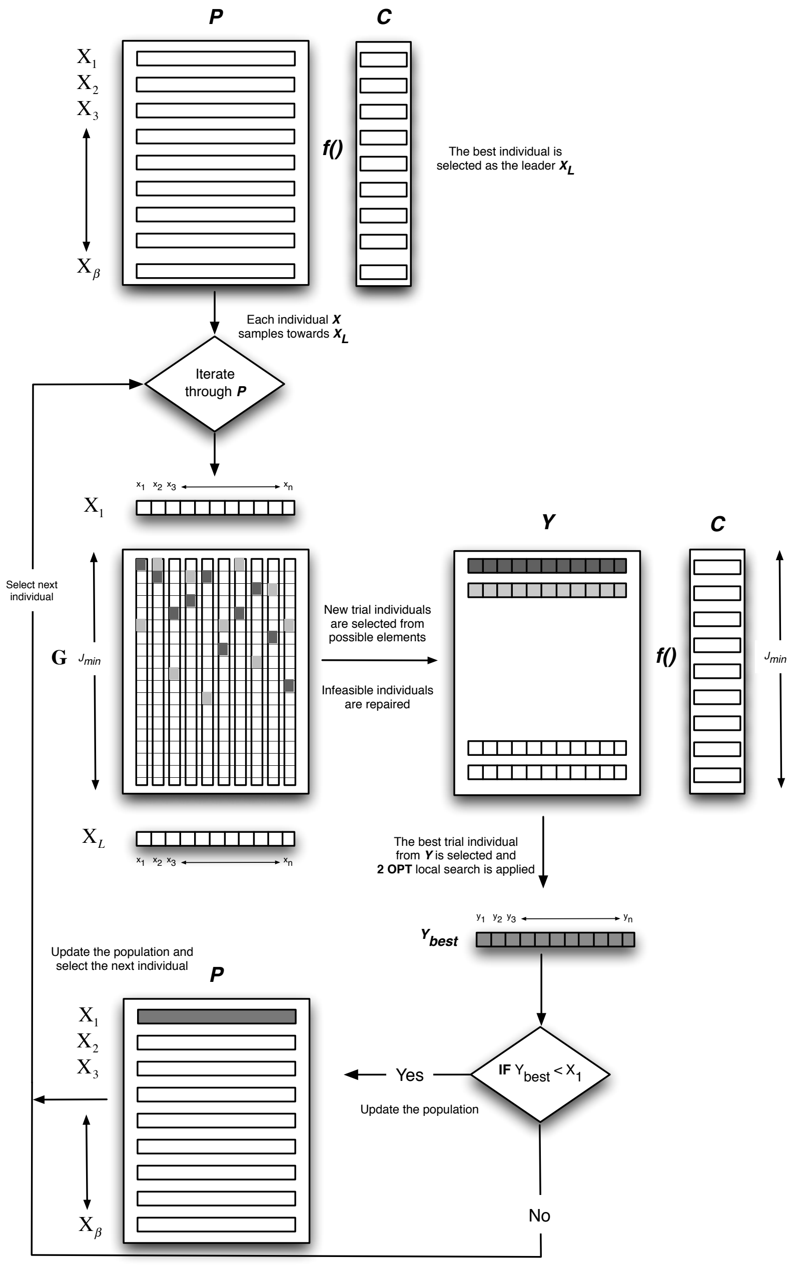

P
C
X1
X2
X3
Xβ
f()
The best individual is selected as the leader XL
Each individual X samples towards XL
Iterate through P
x1 x2 x3 xn
X1
Y
C
Select next individual
G
Jmin
New trial individuals are selected from possible elements
Infeasible individuals are repaired
f()
Jmin
XL
The best trial individual from Y is selected and 2 OPT local search is applied
y1 y2 y3 yn
Ybest
Update the population and select the next individual
P
Yes
IF Ybest < X1
Update the population
No

Figure 2.15: DSOMA schematic.

3 Combinatorial Optimisation Problems

3.1 Introduction

It is first necessary to initially define what a combinatorial problem is. In a combinatorial problem, 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 are 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 [3].

3.2 Permutative Flow Shop Scheduling

In many manufacturing and assembly facilities, a number of operations have to be done on every job. Often these operations have to be done on all the jobs in the same order implying the jobs have to follow the same route. The machines are assumed to be set up in series and the environment is referred to as a *flow shop* (FSS) [26].

The basic representation of the FSS problem as given by [26] is: **Flow Shop** *Fm*: There are *m* machines in series. Each job has be processed in each one of the *m* machines. All the jobs have to follow the same route (i.e., they have to processed on Machine 1, and then on Machine 2, etc). After completing on one machine, a job joins the queue at the next machine. Usually all jobs are assumed to operate under the *First In First Out (FIFO)* discipline - that is a job cannot "pass" another while waiting in a queue. Under this effect the environment is refereed to as a *permutative* flow shop. the general syntax of this problem as described in the triplet format $\alpha|\beta|\gamma$, is given as

$$Fm\,|Perm\,|C_{\max}$$

The first field denotes the problem being solved, the second field the type of problem (in this case permutative) and the last field denotes the objective being under investigation, which is the makespan (total time taken to complete the job).

In order to understand the flow shop problem, assume a permutation schedule $j_1, \ldots, j_n$, where *n* is the number of jobs in the schedule for a m machine flow shop. The subscript *j* refers to a job while the subscript *i* refers to a machine. If a job requires a number of processing steps or operations, then the pair (i, j) refers to the processing step or operation of job *j* on machine *i*. The processing time of each job *j* on each machine *i* can be given as $p_{i,j}$. Subsequently, the completion time of a job j_k at machine *i* can be computed through a set a recursive equations:

$$C_{i,j_1} = \sum_{l=1}^{i} p_{l,j_1} \quad i = 1, \ldots, m \tag{3.1}$$

$$C_{1,j_k} = \sum_{l=1}^{k} p_{1,j_1} \quad k = 1, \dots, n \tag{3.2}$$

$$C_{i,j_k} = \max\left(C_{i-1,j_k}, C_{i,j_{k-1}}\right) + p_{i,j_k} \quad i = 2, \dots, m; k = 2, \dots, n \tag{3.3}$$

The value of the makespan under a given permutation schedule can also be computed by determining the *critical path* in a directed graph corresponding to the schedule.

For a given sequence $j_1, \dots, j_n$, the graph is constructed as follows: For each operation of a specific job j_k on a specific machine *i*, there is a node (i, j_k) with the *processing time* for that job on that machine. Node (i, j_k), $i = 1, \dots, m-1$ and $k = 1, \dots, n-1$, has arcs going to nodes $(i+1, j_k)$ and (i, j_{k+1}). Nodes corresponding to machine *m* have only one outgoing arc, as do the nodes in job j_n. Node (m, j_n), has no outgoing arcs as it is the terminating node and the total weight of the path from first to last node is the makespan for that particular schedule [26]. A schematic is given in Figure 3.1.

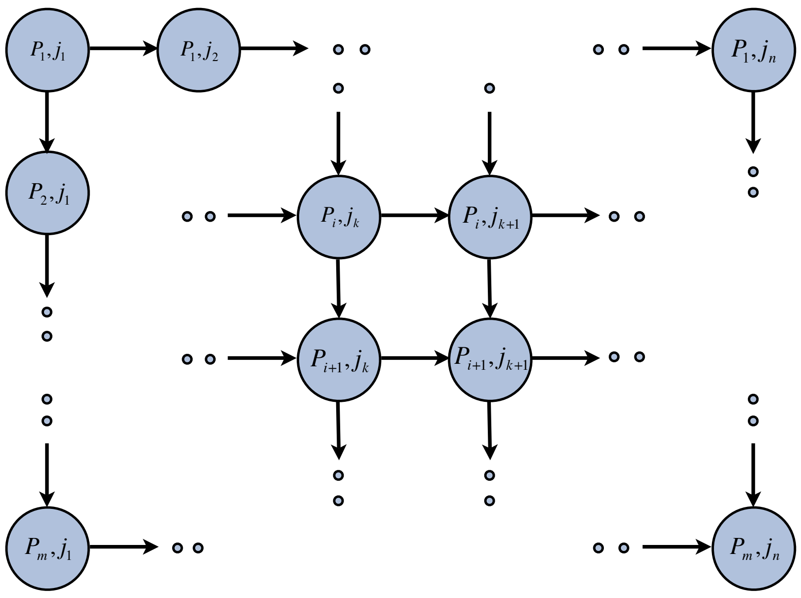


Figure 3.1: Directed graph representation for $Fm\,|Perm\,|C_{\max}$

3.3 Flow Shop Scheduling example

A brief example is presented here in order to show the FSS problem. Consider a 5 job and 4 machine environment with the following processing time as given in Table 3.1.

Assuming a schedule of jobs as 1, 2, 3, 4 and 5, the completion time of the job on the machine can now be given as in Table 3.2. Using equation (3.1), the first job j_1 in the schedule, in this case 1, is processed in all the machines sequentially and the completion

Table 3.1: Sample processing time

jobs	j_1	j_2	j_3	j_4	j_5
p_{1,j_k}	5	5	3	6	3
p_{2,j_k}	4	4	2	4	4
p_{3,j_k}	4	4	3	4	1
p_{4,j_k}	3	6	3	2	5

Table 3.2: Sample completion time for the FSS problem

jobs	j_1	j_2	j_3	j_4	j_5
C_{1,j_k}	5	10	13	19	22
C_{2,j_k}	9	14	16	23	27
C_{3,j_k}	13	18	21	27	28
C_{4,j_k}	16	24	27	29	34

time of the job on each machine can be computed as $C_{1,1}$ = $p_{1,1}$ = 5, $C_{2,1}$ = $C_{1,1}$ + $p_{2,1}$ = (5 + 4) = 9, $C_{3,1}$ = $C_{2,1}$ + $p_{3,1}$ = (9 + 4) = 13 and $C_{4,1}$ = $C_{3,1}$ + $p_{4,1}$ = (13 + 3) = 16.

Equation (3.2) computes the completion time of all the jobs in the schedule on the first machine as $C_{1,2}$ = $C_{1,1}$ + $p_{1,2}$ = 5 + 5 = 10, $C_{1,3}$ = $C_{1,2}$ + $p_{1,3}$ = 10 + 3 = 13, $C_{1,4}$ = $C_{1,3}$ + $p_{1,4}$ = 13 + 6 = 19 and $C_{1,5}$ = $C_{1,4}$ + $p_{1,5}$ = 19 + 3 = 22.

All subsequent jobs are now processed on all subsequent machine (as job 1 has finished on all the machines and the first machine has processed all the jobs). The processing time of the second job in the schedule j_2, is this case job 2, can be calculated using equation (3.3) on all the remaining machines as, $C_{2,2} = \max\left(C_{1,2}, C_{2,1}\right)$ + $p_{2,2}$ = max(10,9) + 4 = 14, $C_{3,2} = \max\left(C_{2,2}, C_{3,1}\right)$ + $p_{3,2}$ = max(14,13) + 4 = 18 and $C_{4,2} = \max\left(C_{3,2}, C_{4,1}\right)$ + $p_{3,2}$ = max(18,16) + 6 = 24. Sequentially, the rest of the job can be processed on the remaining machines. The makespan, in this instance is the completion time of the last job in the schedule (5) on the last machine (4), which in this example is 34.

3.4 Flow Shop Scheduling with Blocking

Consider *m* machines in series with *zero* intermediate storage between successive machines, which have to process *n* jobs. If a given machine finishes the processing of any given job, the job cannot proceed to the next machine while that machine is busy, but must remain on that machine, which therefore remains *idle*. This phenomenon is refereed to as **blocking** [26].

In this section, only flow shops with zero intermediate storage are considered (FSSB), since any flow shop with positive (but finite) intermediate storage between machines can be modelled as a flow shop with zero intermediate storage. This is due to the fact that the storage space capable of containing one job may be regarded as a machine on which the processing time of all machines is equal to zero.

Pinedo [26] has defined the problem of minimising the makespan in a flow shop with zero intermediate storages is referred to in what follows as:

$$Fm\,|block\,|C_{\max}$$

Let $D_{i,j}$ denote the time that job *j* actually departs machine *i*. Clearly $D_{i,j} \geq C_{i,j}$. Equality holds that job *j* is not blocked. The time job *j* starts its processing at the first machine is denoted by $D_{0,j}$. The following recursive relationship hold under the job sequence $j_1, \ldots, j_n$:

$$D_{i,j_1} = \sum_{l=1}^{i} p_{l,j_1} \quad i = 1, \ldots, m \tag{3.4}$$

$$D_{i,j_k} = \max\left(D_{i-1,j_k} + p_{i,j_k}, D_{i+1,j_{k-1}}\right) \quad i = 2, \ldots, m; k = 2, \ldots, n \tag{3.5}$$

$$D_{m,j_k} = D_{m-1,j_k} + p_{m,j_k} \tag{3.6}$$

The makespan can also be calculated by determining the critical path in the directed graph. In this graph, node (i, j_k) is the departure time of job j_k from machine *i*. In contrast with permutative flow shop, in the graph the arcs, rather than the nodes, have weights. Node (i, j_k), $i = 1, \ldots, m-1; k = 1, \ldots, n-1$, has two outgoing arcs; one arc goes to node $(i+1, j_k)$ and has a weight or distance p_{i+1,j_k}, the other arc goes to node $(i-1, j_{k+1})$ and has weight zero. Node (m, j_k) has only one outgoing arc to node $(m-1, j_{k+1})$ with zero weight. Node (i, j_n) has only one outgoing arc to node $(i+1, j_n)$ with weight p_{i+1,j_n}. Node (m, j_n) has no outgoing arcs. The $C_{\max}$ under sequence $j_1, \ldots, j_n$ is equal to the length of the maximum weight path from node $(0, j_1)$ to node (m, j_n).

The directed graph is given in Figure 3.2.

3.5 Flow Shop Scheduling with Blocking example

A brief example is presented here in order to show the FSSB problem. Consider the same problem as in the previous example in section 3.3, of a 5 job and 4 machine environment with the following processing time as given in Table 3.1. The job schedule can again be represented as 1, 2, 3, 4 and 5.

In this case, the departure time of each job on each machine can be given as in Table 3.3. Using equation (3.4), the departure time of the first job on all the machines can be calculated as $D_{1,1}$ = $p_{1,1}$ = 5, $D_{2,1}$ = $D_{1,1}$ + $p_{2,1}$ = 5 + 4 = 9, $D_{3,1}$ = $D_{2,1}$ + $p_{3,1}$ = 9 + 4 = 13 and $D_{4,1}$ = $D_{3,1}$ + $p_{4,1}$ = 13 + 3 = 16.

The departure time of the second job in the schedule j_2 (2 in this example), can be calculated using equation (3.5) as $D_{1,2} = \max\left(D_{1,1} + p_{1,2}, D_{2,1}\right)$ = max(5 + 5, 9) = 10, $D_{2,2} = \max\left(D_{1,2} + p_{2,2}, D_{3,1}\right)$ = max(10 + 4, 13) = 14, $D_{3,2} = \max\left(D_{2,2} + p_{3,2}, D_{4,1}\right)$ = max(14 + 4, 16) = 18. The departure time of j_2 on the last machine can be calculated

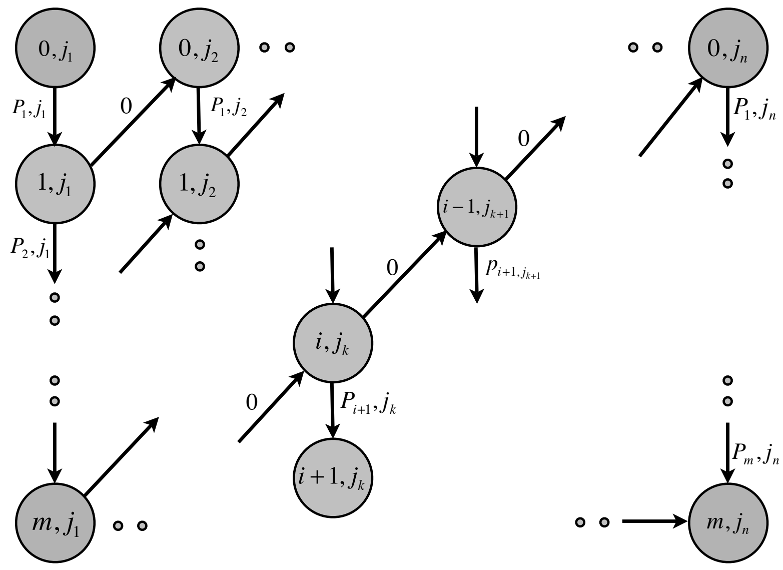


Figure 3.2: Directed graph representation for $Fm\ |block\ |C_{\max}$

Table 3.3: Sample departure time for the FSS with blocking problem

jobs	j_1	j_2	j_3	j_4	j_5
D_{1,j_k}	5	10	14	20	24
D_{2,j_k}	9	14	18	24	28
D_{3,j_k}	13	18	24	28	30
D_{4,j_k}	16	24	27	30	35

using equation (3.6) as $D_{4,2} = D_{3,2} + p_{4,2}$ = 18 + 6 = 24. Sequentially, the reminder of the departure time of the remaining schedule can be thus calculated. The makespan in this case is again the completion time of the final job (5) on the final machine (4), which is 35.

3.6 Flow Shop Scheduling with No Wait

The third variant of flow shop is also the most challenging and practical [26]. Consider a flow shop with zero intermediate storage (FSSNW) subject to different operating procedures. A job, when it goes through the system, is not allowed to *wait* at any machine. For this process, all subsequent machines have to be *idle*, at the completion of the job on a machine upstream. This is the *opposite* to the *blocking* case where the jobs are pushed down by machines upstream. In this case the jobs are ***pulled*** down the line by machines which have become idle. This constraint is refereed to as the **no-wait** constraint, and minimising the makespan in such a flow shop is referred to as the

$$Fm\,|nwt\,|C_{\max}$$

Among all types of scheduling problems, FSSNW owns lots of important applications in different industries such as chemical processing [27], food processing [28], concrete ware production [29], and pharmaceutical processing [30] amongst others.

For the computational complexity of the FSSNW scheduling problem, [31] proves that it is strongly NP-complete. Therefore, only small size instances of the no-wait flowshop problem can be solved with reasonable computational time by exact algorithms.

The following notations are used to formulate the FSSNW problem: as before assume *n* as number of jobs to be scheduled, *m* as the number of machines in the no-wait flowshop, $t_{i,j}$ as the processing time for the i^{th} job on the j^{th} machine, $d_{i,k}$ as the minimum delay on the first machine between the start of job *i* and job *k* due to the no-wait constraint, $[i]$ as the job processed in position *i* , $C_{[i]}$ as the completion time of the job processed in position. *TFT* represents the total flow time, i.e. the sum of flow times of all jobs.

The minimum delay time $d_{i,k}$ and completion time $C_{[i]}$ can be calculated as:

$$\begin{aligned}
&d_{i,k} = t_{i,1} + \max_{2\le j\le m}\left(\sum_{p=2}^{j} t_{i,p} - \sum_{p=1}^{j-1} t_{k,p}\right)\\
&C_{[i]} = \sum_{j=1}^{m} t_{[1],j},\\
&C_{[i]} = \sum_{k=2}^{i} d_{[k-1],[k]}+\sum_{j=1}^{m} t_{[1],j}, \qquad i = 2,3,\ldots,n.
\end{aligned} \tag{3.7}$$

All jobs are assumed to be available at time zero, the total flow time can then be given as in (3.8).

$$\begin{aligned}
&TFT = \sum_{i=2}^{n}\left(\sum_{k=2}^{i} d_{[k-1],[k]} + \sum_{j=1}^{m} t_{[1],j}\right) + \sum_{j=1}^{m} t_{[1],j} =\\
&\sum_{i=2}^{n}\sum_{k=2}^{i} d_{[k-1],[k]}+\sum_{i=1}^{n}\sum_{j=1}^{m} t_{[i],j} =\\
&\sum_{i=2}^{n}(n+1-i)d_{[i-1],[i]} + \sum_{i=1}^{n}\sum_{j=1}^{m} t_{i,j}
\end{aligned} \tag{3.8}$$

where $\sum_{i=1}^{n}\sum_{j=1}^{m} t_{i,j}$ is the sum of the processing time of all jobs in all machines [32].

3.7 Flow Shop Scheduling with No Wait example

Consider the same problem as in the flow shop example in section 3.3, of a 5 job and 4 machine environment with the following processing time as given in Table 3.1. The job schedule can again be represented as 1, 2, 3, 4 and 5.

As in the blocking example, the departure time of the first job j_1 on all the machines can be calculated as $D_{1,1}$ = $p_{1,1}$ = 5, $D_{2,1}$ = $D_{1,1}$ + $p_{2,1}$ = 5 + 4 = 9, $D_{3,1}$ = $D_{2,1}$ + $p_{3,1}$ = 9 +

4 = 13 and $D_{4,1}$ = $D_{3,1}$ + $p_{4,1}$ = 13 + 3 = 16. The start time of the first job can be calculated as the process time subtracted from the completion time as $S_{4,1}$ = $D_{4,1}$ - $p_{4,1}$ = 16 - 3 = 13, $S_{3,1}$ = $D_{3,1}$ - $p_{3,1}$ = 13 - 4 = 9, $S_{2,1}$ = $D_{2,1}$ - $p_{2,1}$ = 9 - 4 = 5 and $S_{1,1}$ = $D_{1,1}$ - $p_{1,1}$ = 5 - 5 = 0.

As the start time of the job cannot be more than the departure time of the same job on a previous machine, the time where the machine is freed for processing is added to all the previous jobs. Stated in another way, the departure time of the same job on a previous machine $D_{i-1,j}$ cannot be less that the departure time of a previous job on the current machine $D_{i,j-1}$. If this condition exists, then the difference between these times $(D_{i,j-1} - D_{i-1,j})$ is added to all the *start times* of the current job in all the previous machines.

The second job j_2 can be calculated iteratively as $D_{1,2}$ = $D_{1,1}$ + $p_{1,2}$ = 5 + 5 = 10, $D_{2,2}$ = if($D_{1,2}$ > $D_{2,1}$) + $p_{2,2}$ = if (10 > 9) + 4 = 14, $D_{3,2}$ = if($D_{2,2}$ > $D_{3,1}$) + $p_{3,2}$ = if (14 > 13) + 4 = 18 and $D_{4,2}$ = if($D_{3,2}$ > $D_{4,1}$) + $p_{4,2}$ = if (18 > 16) + 6 = 24. The start time of j_2 can be now calculated in reverse as $S_{4,2}$ = $D_{4,2}$ - $p_{4,2}$ = 24 - 6 = 18, $S_{3,2}$ = $D_{3,2}$ - $p_{3,2}$ = 18 - 4 = 14, $S_{2,2}$ = $D_{2,2}$ - $p_{2,2}$ = 14 - 4 = 10 and $S_{1,2}$ = $D_{1,2}$ - $p_{1,2}$ = 10 - 5 = 5. Now checking the start time j_2 against the departure time of the j_1, it is obvious that all the start times are after the departure time, therefore no adjustment was required.

The third job j_3 can now be initially calculated as $D_{1,3}$ = $D_{1,2}$ + $p_{1,3}$ = 10 + 3 = 13 and $D_{2,3}$ = if ($D_{1,3} \geq D_{2,2}$) + $p_{2,3}$ = if (13 $\geq$ 14) + 2. As shown here, $D_{1,3}$ is in fact less than $D_{2,2}$, therefore the difference between the departure times $(D_{2,2} - D_{1,3})$ = (14 - 13) = 1 is added to all the start times of the same job in the previous machines, in this case $S_{1,3}$ + 1. Consequentially, the departure time of $D_{1,3}$ will now become (13 + 1) = 14, and the departure time of $D_{2,3}$ can now be obtained as $D_{2,3}$ = if ($D_{1,3} \geq D_{2,2}$) + $p_{2,3}$ = if (14 $\geq$ 14) + 2 = 16. Taking the third machine in this sequence, $D_{3,3}$ = if($D_{2,3} \geq D_{3,2}$) + $p_{3,3}$ = if (16 $\geq$ 18) + 3. Again, the condition does not hold, and as before we have to add the time difference $(D_{3,2} - D_{2,3})$ = (18 - 16) = 2 to all the previous machines as $D_{1,3}$ + 2 = 14 + 2 = 16 and $D_{2,3}$ + 2 = 16 + 2 = 18. $D_{3,3}$ can now be calculated as $D_{3,3}$ = if ($D_{2,3} \geq D_{3,2}$) + $p_{3,3}$ = if (18 $\geq$ 18) + 3 = 21. The departure time on the final machine can be computed as $D_{4,3}$ = if ($D_{3,3} \geq D_{4,2}$) + $p_{3,3}$ = if (21 $\geq$ 24) + 3. Once again the condition fails the test, and subsequently, the difference $(D_{4,2} - D_{3,3})$ = (24 - 21) = 3, is added to all the previous departure times of the same job as $D_{1,3}$ + 3 = 16 + 3 = 19, $D_{2,3}$ + 3 = 18 + 3 = 21 and $D_{3,3}$ + 3 = 21 + 3 = 24. The departure time of $D_{4,3}$ can now finally be calculated as $D_{4,3}$ = if ($D_{3,3} \geq D_{4,2}$) + $p_{3,3}$ = if (24 $\geq$ 24) + 3 = 27.

Likewise, the departure time of the remaining jobs is calculated on all the machines, and the departure time matrix can be given as in Table 3.4, with the start time matrix as in Table 3.5. The *TFT* is the sum of all the time of the jobs, which have been completed on all the machines, so in our case it is the sum of the last row of the departure time matrix as $D_{4,1}$ + $D_{4,2}$ + $D_{4,3}$ + $D_{4,4}$ + $D_{4,5}$ (16 + 24 + 27 + 35 + 40) = 142.

Table 3.4: Sample departure time for the FSS with no wait problem

jobs	j_1	j_2	j_3	j_4	j_5
D_{1,j_k}	5	10	19	25	30
D_{2,j_k}	9	14	21	29	34
D_{3,j_k}	13	18	24	33	35
D_{4,j_k}	16	24	27	35	40

Table 3.5: Sample start time for the FSS with no wait problem

jobs	j_1	j_2	j_3	j_4	j_5
S_{1,j_k}	0	5	16	19	27
S_{2,j_k}	5	10	19	25	30
S_{3,j_k}	9	14	21	29	34
S_{4,j_k}	12	18	24	33	35

4 Complex Network Analysis

This chapter describes and analyses the development of complex network analysis for EDE and DSOMA when applied to the FSS, FSSB and FSSNW problems. Both the algorithms EDE and DSOMA were applied pairwise to the three problems. The stock problem of [33], a 20 job by 5 machine problem was used for all experimentation.

To visualise complex networks, the *Adjacency graph* approach is used, where each generation is represented as adjacency matrix. The creation of the Adjacency graph is given in Algorithm 1 for EDE and Algorithm 2 for DSOMA.

It can be observed that the interactions between individuals generated multiple edges. To get one-to-one relationship, all non-zero had to be reset to one. The appendix section contains complex network graphs generated by the different algorithms for the different problems. There is one figure for each attribute and each figure contain the first six generations of the population evolution. Further generations are omitted due to high density of complex network. All the visualisations was conducted on Mathematica 9 software using the social network analysis toolkit [34].

4.1 Analysis of EDE on FSS

4.1.1 Adjacency Graph

Adjacency is the measure of the number of connections between vertices and edges. This give a good overview on the number of linkages or degrees of nodes. The adjacency matrix graphs are shown in Figure A.1.

At the first sight, it can be seen that the first three generations do not use the all individuals in the population. The first generation have 13 unused nodes (4, 11, 16, 18, 32, 53, 55, 56, 59, 61, 75, 85, 88). There are five left unused in the second generation (32, 53, 55, 85, 16) and in the third generation only one last node is not productive (55). By the fourth generation, all nodes are connected to the graph.

4.1.2 Minimal Cut

A minimum k-cut of a graph is a partition of vertices of a graph into *k* disjoint subsets with the smallest number of edges between them. From the graphs in Figure A.2, it can be seen that the minimal cut actually does not change over the generations. From the analysis of the graphs, it can be seen that the minimal cut remains as two, stating that even over the generations, only two partitions of the population is possible. One colour for each partition is used (yellow and red).

4.1.3 Degree Centrality

The degree centrality is defined as the number of edges connected to a specific node. Nodes with high degree are important hubs and thereby distributes the most information flowing through the network. The graphs are given in Figure A.3. The degree of all nodes is increasing each generation. This help to analyse if *stagnation* or *premature convergence* is

```
input  : Adjacency matrix as $n$ x $n$ to 0, where $n$ is PopulationSize
output: Adjacency matrix
for $i \leftarrow 1$ to $Generations$ do
    for $j \leftarrow 1$ to $PopulationSize$ do
        Randomly select three vectors in the population.;
        Using DE crossover, generate the trial vector;
        $fitness_{trial} \leftarrow$ evaluate $fs_{trial}$;
        if $fitness_{trial}$ < $fitness_{parent}$ then
            replace $parent$ vector with $trial$ vector.;
            add one to corresponding vector row $j$ and $Leader$ vector column in the
            Adjacency matrix.;
        end
    end
end
```

Algorithm 1: EDE Adjacency matrix generation

```
input  : Adjacency matrix as $n$ x $n$ to 0, where $n$ is PopulationSize
output: Adjacency matrix
for $i \leftarrow 1$ to $Migrations$ do
    for $j \leftarrow 1$ to $PopulationSize$ do
        for $k \leftarrow 1$ to $IndividualSize$ do
            Generate values between $J_k$ and $Leader_k$
        end
        $trial \leftarrow$ Repair the new trial individuals.;
        $fitness_{trial} \leftarrow$ evaluate $f_{trial}$;
        if $fitness_{trial} < fitness_{individual}$ then
            add one to corresponding vector row $j$ and $Leader$ vector column in the
            Adjacency matrix.;
        end
    end
end
```

Algorithm 2: DSOMA Adjacency matrix generation

occurring the population. The speed of increase of all nodes is not the same for each, as it is clearly seen that some nodes gets higher degree value faster then the other, and these nodes have bigger influence on the information spread. For example these nodes 1, 11, 12, 13, 22, 24, 26, 30 have a very fast growth of degree. As the degree of node grows, the node is moving toward to the centre of the network.

4.1.4 Closeness Centrality

In a connected graph, having natural distance in the vertices, the distance between nodes is measured by the shortest distance. In other words, the inverse of the sum of all distances of a node in a graph represents its closeness as given in Equation 4.1.

$$C_c(i) = \left[\sum_{j=1}^{N} d(i,j) \right]^{-1} \tag{4.1}$$

Closeness tell us if the node is communicating with others, thus it give us overview to the *rate* of distribution of information in the graph. The graphs are given in Figure A.4. In the first two generations, the centrality is distributed on the outer nodes (6, 20, 94, 68, 47, 65, 15, 19). Therefore, the rest of nodes do not communicate with each other (small nodes) and it displays that the development of the network depends on inner nodes. However, after a number of generations, once all the nodes are connected, the closeness centrality of all nodes is getting to the average value, which points out that the all nodes are trying to participate on the solution equally.

4.1.5 Betweenness Centrality

Betweenness centrality (Figure A.5) gives a number how many *shortest* paths between different pair of nodes goes through the specific node. From the observation is seen that nodes with high betweenness are located in the centre of graph and as their betweenness decreasing they are moving to the edge. For example if one can follow node 33, in first 2 generations it's moving to the centre and getting higher betweenness, then in the third generation it reaches the top of the graph and moves out of the centre. The node shown to have the highest betweenness centrality also has the best fitness function.

4.1.6 Katz Centrality

Katz centrality is a measure of node influence on information flow. It measures the number of all nodes that can be connected through a path with the distances nodes having adverse weighting. The graphs are given in Figure A.6. As expected, an early development of the graph can be seen, with a number of nodes connected in the population, as given in degree centrality. High values of katz centrality can be seen in the centre of network, after generation 17 the value of central node become much lower then the value of outer nodes. This means that the network started to search for the better solution from the outer nodes.

4.1.7 Mean Neighbour Degree

The mean neighbour degree is the average neighbour degree (MND) of the given vertices in a graph as given in Figure A.7. From this specific property, the network is shown to grow more or less proportionally over the generations, with all nodes playing a role in its development.

4.1.8 k-Clique

A k-Clique of a graph is a maximal subset S such that geodesic distance between every pair of vertices in the set is less than or equal to k (set to 2 in this case). That means no two nodes in the set can be more than k steps away from each other. The graphs are given in Figure A.8. From the graphs, it can be seen that the S increases from one node to three nodes over the generations. Also, the position of S changes within the network.

4.1.9 k-Plex

A k-Plex is a maximal subgraph with the following property: each vertex of the induced subgraph is connected to at least n-k other vertices, where n is the number of vertices in the induced subgraph. Here, $k = 2$ and $n = 3$ is used. The graphs are given in Figure A.9.

4.1.10 k-Club

A k-Club of a graph is a subset S of the vertex set, which induces a subgraph of diameter k with the following property: each vertex of the induced subgraph is connected to at least n-k other vertices, where n is the number of vertices in the induced subgraph. Here $n = 2$ have been used. The graphs are given in Figure A.10. K-Club is an important attribute used to analyse dependencies of the network, and to isolate non-productive nodes. Therefore, it is seen that the subset S increases over the generations, implying the growth of the network over the generations.

4.1.11 k-Clan

A k-Clan of a graph is a *k-clique* in which the subgraph induced by subset S has diameter less than or equal to k. So a subset to be a k-Clan-

1. should be a k-clique.
2. all nodes are connected by a path less than or equal to k.

$k = 5$ is used in the experiments. The graphs are given in Figure A.11 and it can be seen that there is an increase of interconnectivity of the network. In fact all nodes are joined over the generations, therefore all nodes share or have had an influence in the generation of the network.

4.1.12 Community Graph Plot

The community graph plot attempts to draw all the vertices grouped into communities, for better representation. In this representation, the *hierarchical* mode of graph representation is presented as given in Figure A.12. The community plot is simply to decipher. At each generation, nodes are grouped and generally eight communities are seen. As the network becomes more dense over the generations, the community graph plot actually decreases, and from the experimentation, reduces to five large groups at the termination of the experiment.

4.2 Analysis of EDE on FSSB

4.2.1 Adjacency Graph

For the adjacency graph for the FSSB problem, the first generation shows five unused nodes (22, 46, 51, 90, 16). However, by the second generation all the nodes are connected to the graph as shown in Figure A.13. Therefore, the growth of this network appears to be much faster than the in case with FSS.

4.2.2 Minimal Cut

The minimal cut of graphs in Figure A.14 shows two separated partitions as was the case in the FSS problem. The difference is that the first two generations have more nodes in the second partition. The number of partitions remains the same through the generations.

4.2.3 Degree Centrality

In the first three generations the network have many hubs which are located in the centre. Therefore, a faster grow of the interconnection can be assumed. The degree centrality of FSSB networks is given on Figure A.15.

4.2.4 Closeness Centrality

The outer nodes from the first generation (26, 41, 71, 76, 80, 97, 100) have a higher closeness then the rest of the population, which implies that they are communicating with other nodes near the centre for to exchange data in order to improves their fitness function. On the other hand, nodes with very low or zero values of closeness do not improve or share their information with others. They have more incoming edges than outgoing. The graphs are given in Figure A.16.

4.2.5 Betweenness Centrality

From figure A.17, it can be seen that only few nodes from each generation have higher betweenness centrality. These nodes appear to gather near the centre of graph.

4.2.6 Katz Centrality

This is a similar situation to the one from FSS, where the nodes from the centre have high value and the edge nodes have lower value. Then after 13 generations the situation changed in the same way as with the FSS problem. The inner nodes were observed to distribute much less amount of information then outer nodes. The graphs are given in Figure A.18.

4.2.7 Mean Neighbour Degree

In the Figure A.19 the evolution of neighbour degree can be visualised. At the early stages the outer nodes are bigger than the inner nodes, which means that their neighbours have higher degree. Therefore the network is growing from the inside to outside. After next few generations the MND is spread more or less equally.

4.2.8 k-Clique

The graphs of k-Clique are given in Figure A.20. The number of members in k-Clique increase from three in first generation to 92 in generation 100. Two or three nodes are added to the k-Clique each generation, whereby after many generations the number of nodes increase only by one or stagnate for a few generations.

4.2.9 k-Plex

It was observed that the number of 2-plexes containing 3 nodes decreased from 100 to 0, and from generation 43 of the iteration, it stagnated at 0. The graphs are given in Figure A.21.

4.2.10 k-Club

K-club of FSSB network grows much faster than FSS. All node joined the k-Club after 25 generations. The graphs are given in Figure A.22.

4.2.11 k-Clan

After the sixth generation all nodes were joined to k-clan and then all nodes were shown to participate on the network evolution. $k = 5$ was used for this experiment. The graphs are given in Figure A.23.

4.2.12 Community Graph Plot

The hierarchical representation is given in Figure A.24. The generations starts from eleven communities but right in the first generation this number shrinks to seven. As the evolution process moves the number of communities oscillate between five and seven.

But the seventh group do not survive to the later generation as other larger groups encompass smaller communities. In later stages the number of communities decrease to four.

4.3 Analysis of EDE on FSSNW

4.3.1 Adjacency Graph

As observed in previous cases of FSS and FSSB, not all individuals are connected to the entire network in the first generation. In the first generation seven nodes remain unconnected and in the second generation only one node is left isolated. But what was not seen in the previous cases is the fact that three out of seven unconnected nodes started cooperating with each other in the first generation (fig. A.25).

4.3.2 Minimal Cut

As in the previous cases, the minimal cut (Figure A.26), show two separated partitions. The difference is that the first two generations have more nodes in the second partition. In this time the partitions look similar to FSS. Even the number of partitions don't change throughout the generations.

4.3.3 Degree Centrality

From the Figure A.27 it can be observed that the evident presence of some higher degree nodes. The early spread of these hubs is more or less equal on the whole network. As the evolution continues these nodes are moving towards the centre.

4.3.4 Closeness Centrality

The distribution of nodes with high closeness is not so strongly tied to the edges of the network as it was in the previous cases. In the later stages it is possible to see the convergence to the average value. From generation 40 onwards it is seen that central nodes have much higher closeness. The graphs are given in Figure A.28.

4.3.5 Betweenness Centrality

The situation on Figure A.29 looks similar to the FSSB problem when there are only few node with very high betweenness. These nodes appears to have high degree and closeness as well.

4.3.6 Katz Centrality

The katz centrality given in the Figure A.30, shows that the number of nodes which distribute more information is growing in the centre. Then in later generations the network appears to follow the same order as the previous problems (FSS, FSSB). In the later generations the information flow moves from the inside nodes to the outside.

4.3.7 Mean Neighbour Degree

The mean neighbour degree is the average neighbour degree of the given vertices in a graph as given in Figure A.31.

4.3.8 k-Clique

Again the number of nodes joining the k-clique in Figure A.32 is growing and the rate of growth is much faster compared to FSS and FSSB.

4.3.9 k-Plex

2-plexes with 3 nodes is present in all generations from 1 to 23. Beyond this there is no more 2-plex. k-plex is also changing position which can be seen in Figure A.33.

4.3.10 k-Club

The rate of nodes joining the k-club is again higher, as in the previous case. By the 16 generation k-club already contains all nodes interconnected. The graphs are given in Figure A.34.

4.3.11 k-Clan

As in the previous case all nodes joined the k-clan and this occurred in the sixth generation. It can be seen in Figure A.35, that there is an increase of interconnectivity of the network.

4.3.12 Community Graph Plot

The community starts from 11 different groups where one group is created from three nodes and it is not connected to the rest of the network. Then the number of communities decreases each generation stabilising at five communities, as can be observed on Figure A.36.

4.4 Analysis of DSOMA on FSS

4.4.1 Adjacency Graph

The adjacency graph in Figure A.37 gives a nice overview on DSOMA. As expected the star topology is created with central node as a leader. In the first generation, six nodes are unused (33, 49, 60, 88, 92, 23). All nodes except one are joined to the leader in the next generation whereas the joining of last node takes nine generations.

4.4.2 Minimal Cut

The minimal cut of graphs in Figure A.38, show two separated partitions as it was in EDE FSS case. The number of partitions remains the same through the generations.

4.4.3 Degree Centrality

The degree centrality of the networks is give in Figure A.39 where, as expected the leader has the highest degree and the rest of the nodes have degree of one or $\max$ *number of current leaders*. From the point of leaders view, the number of following nodes is clearly seen. Likewise, when observing a node, it is also possible to see the linkages to different leaders it is following.

4.4.4 Closeness Centrality

The graphs for closeness centrality are given in Figure A.40. It can be observed that the network is split between two leaders after the first generation, which gravitate towards the centre.

4.4.5 Betweenness Centrality

All nodes except the leaders have $BC = 0$, as observed from the Figure A.41. It can also be seen that the betweenness decreases over the generations until a new leader appears and it's betweenness centrality is higher then the previous leaders.

4.4.6 Katz Centrality

The graphs of Katz centrality given in Figure A.42 give us an overview on the distribution of information. It can be seen that as the node becomes the leader, the amount of information it is distributing increases proportionally.

4.4.7 Mean Neighbor Degree

The higher weighted nodes on Figure A.43 have a high degree neighbours which encompass only the following nodes.

4.4.8 k-Clique

The k-Clique consist only the leaders because they are the only nodes which are connected to themselves and other nodes. The graphs are given Figure A.44.

4.4.9 k-Plex

There will always be many 2-plexes with 3 nodes because all nodes are connected through the leaders. The graphs are given in Figure A.45.

4.4.10 k-Club

The number of members in k-Club increase over the generation as the interconnectivity increases over time. The graphs are given in Figure A.46.

4.4.11 k-Clan

Since the k-Clan is an inducted subgraph, its members can be only leaders. So in this case in order to visualise the graph is to only find the leaders by k-Clan (Figure A.47).

4.4.12 Community Graph Plot

Hierarchical representation of communities give a nice geographic overview on what is happening in the network. Figure A.48 shows that the leaders obtain nodes which are interconnected to them. The number of nodes is not static, and not all nodes are connected to the community. Generally, the evolution of leaders occurs within the communities, so there is general static overlay throughout the generations.

4.5 Analysis of DSOMA on FSSB

4.5.1 Adjacency Graph

In the first generation there are twelve nodes unused, which decreases to three in the second generation and all nodes are connected by the fourth generation. The graphs are given in Figure A.49.

4.5.2 Minimal Cut

The minimal cut of graphs in Figure A.50, show two separated partitions as it was in DSOMA on FSS. Also the node 99 is alone in second partition. The number of partitions remains the same through the generations.

4.5.3 Degree Centrality

The degree centrality of DSOMA on FSSB networks on Figure A.51 shows the first two leaders (12, 42) keep their betweenness higher then their followers.

4.5.4 Closeness Centrality

As in the previous case, the degree of closeness centrality starts at one in the first generation, and increases to two in the second generation and remains at two throughout. The graphs are given in Figure A.52.

4.5.5 Betweenness Centrality

Central nodes have more control of the information flow than the outer nodes as given in Fig. A.53. The two nodes oscillate with respect to the rate of information exchange, as show by the depth of the central nodes.

4.5.6 Katz Centrality

The Katz centrality figures are given in Figure A.54. As in the previous case of FSS problem, the centrality increases to two in the second generation and converges to the centre.

4.5.7 Mean Neighbour Degree

The mean neighbour degree given in Figure A.55 shows the oscillation of the network over the two central nodes. It can be seen that the distribution is almost even by the fifth iteration.

4.5.8 k-Clique

As in the previous case, the k-Clique shows the interconnection of the nodes to the leaders. The graphs are given in Figure A.56. Another interesting feature is that the central leader nodes converge due to the level of interconnection.

4.5.9 k-Plex

Here $k = 2$ and $n = 3$ have been used. The graphs are given in Figure A.57 and it can be seen that the dominant leader exhibits three vertices, with one self-looping vertices and from the fifth generation, the all the nodes are connected to the network.

4.5.10 k-Club

As in the previous case the k-Club attributes outline the growth of the network over the generations. The graphs are given in Figure A.58. As is typical in a star-network, the dependencies of the network, especially for $n = 2$ can be easily linked if two nodes share a common leader.

4.5.11 k-Clan

The graphs are given in Figure A.59 and it can be seen that there is an increase of interconnectivity of the network as in the case of the previous problem. Another important feature is the sub-graph connection between the leaders.

4.5.12 Community Graph Plot

Hierarchical graph representation is given in Figure A.60. As in the previous problem, the number of communities converge to two after the fifth generation. In the first generation, only one major community is visible, which then splits into two in the second generation. These two communities absorb all other isolated nodes.

4.6 Analysis of DSOMA on FSSNW

4.6.1 Adjacency Graph

In the first generation there are seven nodes unused (59, 63, 77, 79, 86, 95, 32), which decreases to one node in the third generation and all nodes are connected by the sixth generation. The grow of this network appears to be much faster than the in case with FSS problem. Another interesting issue is that the first generation has one leader, which increases to two in the second generation, three in the fourth generation and four in the sixth generation.

4.6.2 Minimal Cut

The minimal cut of graphs in Figure A.62, show two separated partitions in the second generation, which remains throughout even though the number of leaders increases over the generations.

4.6.3 Degree Centrality

The degree centrality of FSSNW networks on Figure A.63 appears to have much more hubs and they are located in the centre. The number of hubs are corresponding to the number of leaders in the population.

4.6.4 Closeness Centrality

The graphs of closeness centrality are given in Figure A.64. It can be observed that the centrality converges to the four leaders in the sixth generation, therefore the rate of information flow in centralised.

4.6.5 Betweenness Centrality

Central nodes have more control of the information flow than the outer nodes (fig A.65). By observing the depth of the nodes, it can be stated that two leaders collating in the centre have better fitness values than the other two leaders, which have been pushed to the edges.

4.6.6 Katz Centrality

Katz centrality in this case is a mirror of the degree centrality, and reinforces the arguments presented before. The graphs are given in Figure A.66.

4.6.7 Mean Neighbour Degree

The mean neighbour degree is the average neighbour degree of the given vertices in a graph as given in Figure A.67. The interesting point here is that even though two leaders evolved early, more leaders were able to get rapid connections to the network, increasing the neighbourhood degree.

4.6.8 k-Clique

The numbers of k-Clique's increases over the generations, as shown through the red connections in the Figure A.68. As previously described, all the leaders exhibit k-Clique's over the generations.

4.6.9 k-Plex

K-Plex throughout the generations can be seen as three (as $k = 2$ and $n = 3$). however, it becomes easier to isolate the best leader, as the k-Plex in centralised to it. From the graphs, even through four leaders are visible, it is easy to isolate the best performing leader as given in Figure A.69.

4.6.10 k-Club

The growth of the network is easily seen through the k-Club analysis. As in the previous case, the number of subgraphs increases proportionate to the leaders. From Figure A.70, is can be seen that all leaders are connected through dependencies leading to the growth of the network.

4.6.11 k-Clan

The k-Clan parameter of $k = 5$ shows a growth of the interconnection of the network. What is important is that all leaders are shown to be interconnected to the network as given in Figure A.71.

4.6.12 Community Graph Plot

The community graph plot for the DSOMA FSSNW gives a very interesting outline of the network. The community starts from one in generation one, and increases to two in generation two. The communities then increase to three in generation three, and even though a fourth leader appears in generation six, only three communities exist. This gives a clear indication that even though more leaders emerge, this does not translate to more communities in the population as given in Figure A.72.

5 Conclusion

This thesis aims to answer a few important issues. These can be summed up as the following:

1. Do different evolutionary algorithms with different paradigms generate complex networks?
2. Do different problem classes affect the complex network generation?
3. Can meaningful attributes be obtained using complex network tool from evolutionary algorithms.
4. What benefits can evolutionary algorithms have from complex network analysis

In order to answer these questions, two different EA's of EDE and DSOMA were used for complex network generation. The problems selected were the permutative flowshop scheduling problem, flowshop with blocking constraint problem and the flowshop with no-wait constraint problem. For all the problems, the objective was to minimise the *makespan* (total time). These problems were selected due to their NP-Hardness and the fact that they are widely applied in industrial applications.

Each algorithm was applied to the three different problems, leading to six different complex networks. For each network, twelve different parameters were measured. Therefore, in total 72 different analysis were conducted for broad generalisation of this thesis.

From the analysis of the networks, the following can be summarised that:

1. Both EDE and DSOMA generated complex network structure for all the three problems. This behaviour is seen as the expansion of the population in terms of improving fitness, clustering due to fitness convergence and community generation due to interaction.
2. Different paradigm based EA's generate different network structures. This is apparent through the interaction of the population. EDE appears more diffused in the generations, whereas DSOMA is a star-based topological system.
3. Different problem classes do have an impact on the complex network generation, with higher constraint based problems leading to faster integration of the network as seen in EDE FFSB and DSOMA FFSB problems.
4. Some of the more meaningful attributes that can be measured are that:
   (a) not all individuals in the population interact at each generation. However, after repeated experimentation, it was clear that at the completion of the generations, all individuals were connected in the network. Therefore, it is clear that population size has an impact on the selection of new individuals. The different centralities show that a number of hubs, or connection point with

more nodes appear. These are the optima regions in the fitness landscape of the problem.

(b) centrality analysis gives an overview of the central nodes the population. Stagnation or convergence can be easily identified using these types of analysis, where the frequency of the nodes and the depth of interconnection can be measured. From the experiments, it was easy to identify that both EDE and DSOMA exhibit the level of centrality between the "best" performing nodes, and the rest of the population.

(c) clique analysis gives an overview of the number of sub-connections within the populations. These subsets can give an overview of the density of the population and the depth of the connections. In the experimentations, it was observed that k-Club and k-Clan increase during generations for both DSOMA and EDE. The maximal subset was kept at two (k = 2) for all clique analysis, in order to gauge the interaction rate of the population.

(d) the community plot gives a hierarchical overview of the population. The individuals appear clustered throughout the generations, and when local optima stagnation occur, the communities shrink as the connections and individuals stagnate in specific regions of the fitness landscape. This is an important gauge of the viability of the population.

It can be seen that an EA population can also be analysed through complex network tools, which give an excellent representation of the interconnections and dynamics, thereby shedding more light on better management on control of EA's, especially related to stagnation of the population.

Some of the insights obtained from this research is that evolutionary algorithms in *general* exhibit complex network dynamics, regardless of the problem specifics.

One of the most important attributes gained is that EA's can be analysed during evolution using complex networks, and the following can be proposed:

1. use of *centrality* to identify and isolate the best performing individuals in the population, and therefore increasing their "footprint" in the search space through interconnection,

2. use of *cliques* to analyse the development of the interconnection of the network, and the isolation and possibly replacement of non-productive nodes,

3. use of *community plots* to map the population in terms of its *variance* (uniqueness)

The above mentioned features could be implemented in real-time inside the EA's environment, which could then be used as a *feedback* system for the control of the operating parameters. These could be in the form of adjusting the scaling factor in EDE or PathLength in DSOMA. Another possibility could be the replacement of long term non-productive individuals in the population, with better "designed" individuals.

It could finally be summed that EA's do exhibit complex network behaviour when applied to combinatorial optimisation problems, and the population dynamics can be easily measured using complex network tools, which ultimately leads to a better understanding of the algorithm and problem.

6 References

[1] P. Erdős and A. Rényi, "On random graphs. I," *Publ. Math. Debrecen*, vol. 6, pp. 290–297, 1959.

[2] K. Price, R. Storn, and J. Lampinen, *Differential Evolution - A Practical Approach to Global Optimization*. Springer, Germany, 2005.

[3] G. Onwubolu and D. Davendra, *Differential Evolution: A Handbook for Global Permutation-Based Combinatorial Optimization*. Springer, Germany, 2009.

[4] I. Zelinka, "Soma - self organizing migrating algorithm," in *New Optimization Techniques in Engineering* (O. G. and B. B., eds.), Germany: Springer-Verlag, 2004.

[5] D. Davendra, I. Zelinka, M. Bialic-Davendra, R. Senkerik, and R. Jasek, "Discrete self-organising migrating algorithm for flow-shop scheduling with no-wait makespan," *Mathematical and Computer Modelling*, vol. doi:10.1016/j.mcm.2011.05.029, 2011.

[6] G. Chen, X. Wang, and X. Li, *Introduction to Complex Networks: Models, Structures and Dynamics*. John Wiley & Sons (Asia) Pte Limited Pacific, 2014.

[7] S. N. Dorogovtsev and J. F. Mendes, "Evolution of networks," *Advances in physics*, vol. 51, no. 4, pp. 1079–1187, 2002.

[8] B. Stefan and S. Heinz, *Handbook of Graphs and Networks From the Genome to the Internet*. Wiley-VCH GmgH & Cp. KgaA, Weihheim, 2003.

[9] S. Milgram, "The small world problem," *Psychology Today*, vol. 67, no. 1, pp. 61–67, 1967.

[10] L. Backstrom, P. Boldi, M. Rosa, J. Ugander, and S. Vigna, "Four degrees of separation," in *Proceedings of the 3rd Annual ACM Web Science Conference*, WebSci '12, (New York, NY, USA), pp. 33–42, ACM, 2012.

[11] D. Watts and S. Strogatz, "Collective dynamics of 'small-world' networks," *Nature*, no. 393, pp. 440–442, 1998.

[12] M. Newman and D. Watts, "Scaling and percolation in the small-world network model," *Phys. Rev. E*, vol. 60, pp. 7332–7342, Dec. 1999.

[13] I. Zelinka, D. Davendra, V. Snášel, R. Jašek, R. Šenkeřík, and Z. Oplatková, "Preliminary investigation on relations between complex networks and evolutionary algorithms dynamics," pp. 148–153, 2010. cited By (since 1996)0.

[14] I. Zelinka, "Investigation on relationship between complex networks and evolutionary algorithms dynamics," vol. 1389, pp. 1011–1014, 2011. cited By (since 1996)0.

[15] I. Zelinka, D. Davendra, and L. Skanderova, "Visualization of complex networks dynamics: Case study," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 7291 LNCS, pp. 145–150, 2012. cited By (since 1996)0.

[16] I. Zelinka, L. Skanderová, D. Davendra, R. Senkerik, and Z. Oplatkova, "Evolutionary dynamics and complex networks," pp. 88–93, 2012. cited By (since 1996)0.

[17] K. Táng, X. Lǐ, P. N. Suganthan, Z. Yáng, and T. Weise, "Benchmark Functions for the CEC'2010 Special Session and Competition on Large-Scale Global Optimization," tech. rep., University of Science and Technology of China (USTC), School of Computer Science and Technology, Nature Inspired Computation and Applications Laboratory (NICAL): Héféi, Ānhuī, China, Jan. 8, 2010.

[18] G. Onwubolu and D. Davendra, "Scheduling flow shops using differential evolution algorithm," *European Journal of Operations Research*, vol. 171, pp. 674–679, 2006.

[19] G. Onwubolu, "Optimization using differential evolution," in *Technical Report TR-2001-05*, (IAS, USP, Fiji), Oct. 2001.

[20] D. Davendra and G. Onwubolu, "Enhanced differential evolution hybrid scatter search for discrete optimisation," in *Proc. of the IEEE Congress on Evolutionary Computation*, (Singapore), pp. 1156–1162, Sept 2007.

[21] D. Davendra and G. Onwubolu, "Flow shop scheduling using enhanced differential evolution," in *Proc.21 European Conference on Modeling and Simulation*, (Prague, Czech Rep), pp. 259–264, Jun 2007.

[22] D. Davendra and G. Onwubolu, "Forward backward transformation," in *Differential Evolution: A Handbook for Global Permutation-Based Combinatorial Optimization* (G. Onwubolu and D. Davendra, eds.), Germany: Springer, 2009.

[23] Z. Michalewicz, *Genetic Algorithm + Data Structures = Evolution*. Springer, Germany, 1994.

[24] G. Onwubolu and T. Kumalo, "Optimization of multi-pass turning operations with genetic algorithms," *International Journal of Production Research*, vol. 39, pp. 3727–3745, 2001.

[25] D. Davendra, *Chaotic Attributes and Permutative Optimization*. PhD thesis, Tomas Bata University in Zlin, 2009.

[26] M. Pinedo, *Scheduling: theory, algorithms and systems*. Prentice Hall, Inc., New Jersey, 1995.

[27] C. Rajendran, "A no-wait flowshop scheduling heuristic to minimize makespan," *Journal of the Operational Research Society*, pp. 472–478, 1994.

[28] N. Hall and C. Sriskandarayah, "A survey of machine scheduling problems with blocking and no-wait in process," *Operations Research*, pp. 510–525, 1996.

[29] J. Grabowski and J. Pempera, "Sequencing of jobs in some production system," *European Journal of Operational Research*, pp. 535–550, 2000.

[30] W. Raaymakers and J. Hoogeveen, "Scheduling multipurpose batch process industries with no-wait restrictions by simulated annealing," *European Journal of Operational Research*, pp. 131–151, 2000.

[31] M. Garey and D. Johnson, *Computers and intractability: A guide to the theory of NP-completeness*. Freeman, San Francisco, 1979.

[32] J.-L. Chang, D.-W. Gong, and X.-P. Ma, "A heuristic genetic algorithm for no-wait flowshop scheduling problem," *Journal of China University of Mining and Technology*, vol. 17, no. 4, pp. 582 – 586, 2007.

[33] E. Taillard, "Benchmarks for basic scheduling problems," *European Journal of Operations Research*, vol. 64, pp. 278–285, 1993.

[34] Wolfram, "Wolfram website," 2014. `http://www.wolfram.com/`.

A Graphs of created networks

To view figures in appendix please open the appendix file.