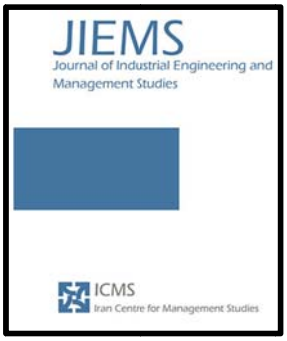
JIEMS
Journal of Industrial Engineering and Management Studies

Vol. 2, No. 1, pp. 41-60

www.jiems.icms.ac.ir



Particle swarm optimization for a bi-objective web-based convergent product networks

R. Hassanzadeh[1], I. Mahdavi[*, 1], N. Mahdavi-Amiri[1]

Abstract
Here, a collection of base functions and sub-functions configure the nodes of a web-based (digital) network representing functionalities. Each arc in the network is to be assigned as the link between two nodes. The aim is to find an optimal tree of functionalities in the network adding value to the product in the web environment. First, a purification process is performed in the product network to assign the links among bases and sub-functions. Then, numerical values as benefits and costs are determined for arcs and nodes, respectively. To handle the bi-objective Steiner tree, a particle swarm optimization algorithm is adapted to find the optimal tree determining the value adding sub-functions to bases in a convergent product. An example is worked out to illustrate the applicability of the proposed approach.

Keywords: Convergent product; Web-based (digital) network; Bi-objective programming; Steiner tree; Particle Swarm Optimization (PSO).

Received: April 2014-11

Revised: May 2014-15

Accepted: July 2014-23

1. Introduction

Convergence in electronics and communications sectors has enabled the addition of disparate new functionalities to existing base functions (e.g., adding mobile television to a cell phone or Internet access to a personal digital assistant, PDA). An important managerial issue for such convergent products (CPs) is determination of new functionalities adding more value to a given base. For example, a manufacturer of PDAs may wonder whether it would be a good idea to add satellite radio to it (i.e., a new functionality incongruent with the base), or whether it would be better to add electronic Yellow Pages (i.e., a new functionality congruent with the functions of a PDA). In addition, determining the significance of the base being primarily associated with utilitarian consumption goals (e.g., a PDA), or with hedonic ones (e.g., an MP3 music player) is important.

Convergent product is similar to product assembly where different parts of a product get together to configure a final product. Thus, a designer (modeler) for assembly, as a convergent product, should be able to specify important features affecting the final product. These features may in turn help optimize the manufacturing process.

[*]Corresponding Author.
[1] Dep. Of Management & Technology, Mazandaran University of Science and Technology, Mazandaran, Iran

For instance, the ability of the assembly modeler to furnish information on interferences and clearances between mating parts is particularly useful. Such information would enable the designer to eliminate interference between two mating parts where it is impractical to provide for an interference based on physical assembly requirements. This activity can be accomplished within the modeling program, thereby averting any loss of productivity that might occur due to interferences on the shop floor. Also, knowledge of mass properties for the entire assembly, particularly the center of gravity, may permit the designer to redesign the assembly based on equilibrium and stability considerations. In the absence of such information, the presence of an elevated center of gravity and the attendant instability would only be detected after physical assembly on the shop floor. Three-dimensional exploded views generated by the assembly modeler can help designers verify whether obvious violations of common design for assembly (DFA) guidelines are present, such as absence of chamfers on mating parts.

Corresponding analyses can be achieved within the framework of the assembly modeler. Additionally, the assembly model may be imported into third-party programs that can perform kinematic, dynamic, or tolerance analysis. Tolerance analysis is quite relevant to the physical assembly process. With the input of the assembly model and other user-supplied information such as individual part tolerances, tolerance analysis programs can check the assembly for the presence of tolerance stacks. Tolerance stacks are undesirable elements in the sense that acceptable tolerances on individual parts are combined to produce an unacceptable overall dimensional relationship, thereby resulting in a malfunctioning or nonfunctioning assembly. Stacks are usually discovered during physical assembly, at which point any remedial procedure becomes expensive in terms of time and cost. Tolerance analysis programs can help the user eliminate or significantly reduce the likelihood of stacks being present.

Based on the results of the tolerance analysis, assembly designs may be optimized by modifying individual part tolerances. Note, however, that tolerance modifications have cost implications; in general, tighter tolerances increase production costs. Engineering handbooks contain tolerance charts indicating the range of tolerances achieved by manufacturing processes such as turning, milling, and grinding. Designers use these tables as guides for rationally assigning part tolerances and selecting manufacturing processes.

A more effective methodology for optimizing product assembly and convergent product is the tree model, whereas the optimization decision is based on a decision tree. One useful tree for assembly modelers as a multiple optimization tool is the Steiner tree.

The Steiner tree problem (STP) is a much actively investigated problem in graph theory and combinatorial optimization. This core problem poses significant algorithmic challenges and arises in several applications where it serves as a building block for many complex network design problems. Given a connected undirected graph $G=(V,E)$, where V denotes the set of nodes and E is the set of edges, along with a weight C_e associated with each edge $e \in E$, the Steiner tree problem seeks a minimum-weight subtree of G that spans a specified subset $N \subset V$ of *terminal nodes*, optionally using the subset $N=V-N$ of *Steiner nodes*. The Steiner tree problem is NP-hard for most relevant classes of graphs (Johnson, 1985).

The Steiner problem in graphs was originally formulated by Hakimi (1971). Since then, the problem has received considerable attention in the literature. Several exact algorithms and heuristics have been proposed and discussed. Hakimi (1971) remarked that an *Steiner minimal tree* (SMT) for X in a network $G=(V,E)$ can be found by enumerating minimum spanning trees of subgraphs of G induced by supersets of X. Lawler (1976) suggested a modification of this algorithm, using the fact that the number of Steiner points is bounded by $|X|-2$, showing that not all subsets of V need to be considered. Restricting NP-hard algorithmic problems regarding arbitrary graphs to a smaller class of graphs will sometimes, yet not always, result in polynomially solvable problems.

Two special cases of the problem, $N = V$ and $N = 2$, can be solved by polynomial time algorithms. When $N = V$, the optimal solution of STP is obviously the spanning tree of G and thus the problem can be solved by polynomial time algorithms such as Prim's algorithm. When $N = 2$, the shortest path between two terminal nodes, which can be found by Dijkstra's algorithm, is exactly the Steiner minimum tree.

A survey of Steiner tree problem was given by Hwang and Richards (1992). Several exact algorithms have been proposed such as dynamic programming technique given by Dreyfuss and Wagner (1971), Lagrangean relaxation approach presented by Beasley (1989), brand-and-cut algorithm used by Koch and Martin (1998). Duin and Volgenant (1989) presented some techniques to reduce the size of the graphs for the GSP. Another approach for the GSP is using approximation algorithms to find a near-optimal solution in a reasonable time.

Some heuristic algorithms have been developed such as Shortest Path Heuristic (SPH) given by Takahashi and Matsuyama (1980), Distance Network Heuristic (DNH) presented by Kou et al. (1981), Average Distance Heuristic (ADH) proposed by Rayward-Smith and Clare (1986) and Path-Distance Heuristic (PDH) presented by Winter and MacGregor Smith (1992). Mehlhorn (1988) modified the DNH to make the algorithm faster. Robins and Zelikovsky (2000, 2005) proposed algorithms improving the performance ratio.

Recently, metaheuristics have been considered to arrive at better methods for finding solutions closer to the optimum. Examples are Genetic Algorithm (GA) (Esbensen, 1995; Kapsalis, et al., 1993), GRASP (Martins et al., 1999) and Tabu search (Ribeiro and Souza, 2000). Although these algorithms have polynomial time complexities, in general, but they cost enormously on large input sets. To deal with the cost issue, some parallel metaheuristic algorithms have been proposed such as parallel GRASP (Martins et al., 1998), parallel GRASP using hybrid local search (Martins et al., 2000) and parallel GA (Fatta et al., 2003).

Here, using the Steiner tree, a bi-objective mathematical model is developed for the convergent product. The remainder of our work is organized as follows. In Section 2, the proposed model of the problem is described and two useful network algorithms are given. Section 3 presents the mathematical model and a Particle Swarm Optimization algorithm. Section 4 works out an experimental study to illustrate the proposed algorithm. We conclude in Section 5.

2. The proposed model

In our proposed product digital network, a group of functionalities are considered for a product. Customers view their opinions for classifying the functionalities into base functions and sub-functions. We make use of this classification in developing our model. The classification procedure is as follows. First, the customer chooses a product in a list of products being produced in a company. The functionalities of the product are viewed in a web page. Then, the customer clicks either function or sub-function for any of the functionalities. Consequently, customer clicks the classify button and observes the classified functionalities in a separate web page. This process is shown in Figure 1.

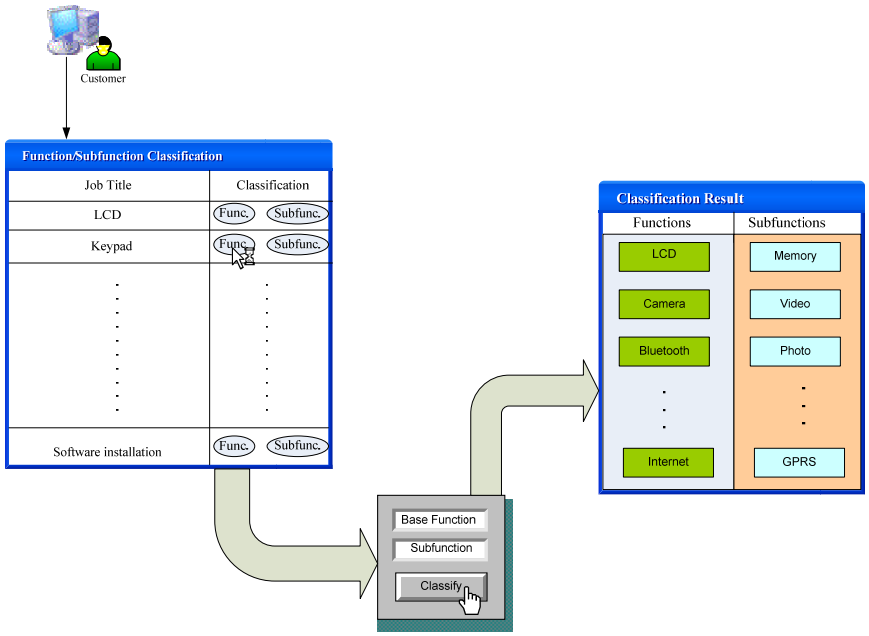


Figure 1: The classification process

Here, we weigh all functionalities (both base functions and sub-functions) considering different significant attributes affecting the value of a product. Therefore, we consider the following mathematical notations.

2.1. Mathematical notations:

i and j	Index for functions and sub-functions;	i and j=1,…,n+m
k	Index for attributes;	k=1,…,p
F_{ijk}	The score of triplet comparison of functions (or sub-functions) with functions (or sub-functions) considering different attributes.	

The three dimension comparison matrix F is shown in Figure 2. Note that customers fill in this matrix using numerical values $F_{ijk} \in [0,1]$.

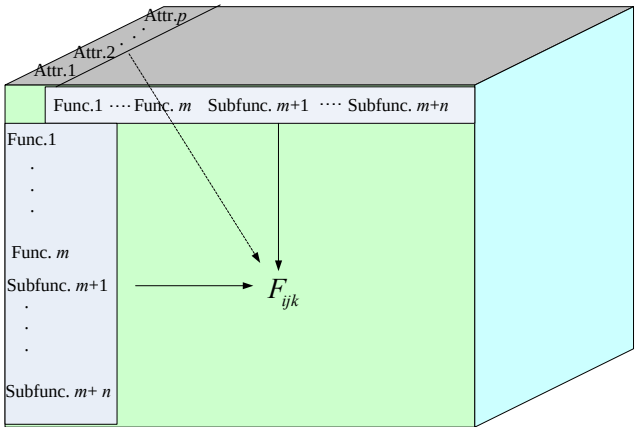


Figure 2: The three dimensional comparison matrix

This matrix is normalized to remove the scales. The normalized values are shown by $\overset{norm}{F}_{ijk}$. A threshold value of θ is considered in a way that the $\overset{norm}{F}_{ijk} \geq \theta$ are chosen to be assigned as links. These links configure a network called purified network as shown in Figure 3.

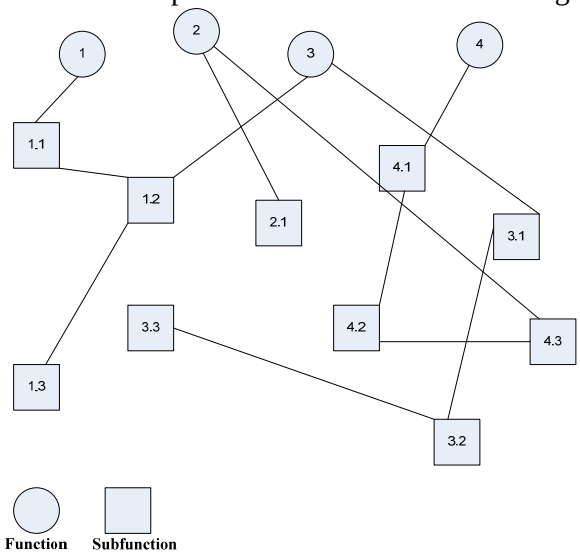


Figure 3: A purified network

Now, using the purified network, we characterize the arcs. To do this, two processes of leveling and clustering are performed. For leveling, we set the base functions at level zero, sub-functions with one outlet to the previous level in level 1, and so on. Thus, an *l* level network is configured. The proposed algorithm is given next.

Algorithm 1: Leveling to configure a leveled network.

Step 0: Set the base functions at level 0. Let *l*=0.
Step 1: **While** sub-functions exist for processing **do**
Find sub-functions with a link to a function (or sub-function) at level *l* and put them in level *l*+1. Let *l*=*l*+1.
End while.
{*l* is the number of levels.}
Step 2: Stop.

The nodes of leveled network are associated with given costs. We are looking for the benefit each link provides. Here, a clustering approach is considered. Clusters are formed as follows: at each level, all sub-functions linked to a single parent is grouped in a cluster. Therefore, clusters consisting different nodes are configured. These clusters are being configured as a new network. The leveling and clustering processes are shown schematically in Figure 4. Later, we apply the Steiner tree methodology to optimize this network. The proposed algorithm for clustering is given below.

Algorithm 2: Clustering of levels in a leveled network.

Step 0: Set each node at level 0 to be a cluster.
Step 1: **For** *i*=1 to *l* **do**
{Form clusters at level *i*}
Cluster all sub-functions at level *i* linked to a single parent at level *i*-1.
Solve a zero/one mathematical program for level *l* (we will discuss the corresponding mathematical program later on).
Perform purification of benefits and costs at level *l* (as discussed later on).

End for
Step 2: Stop.

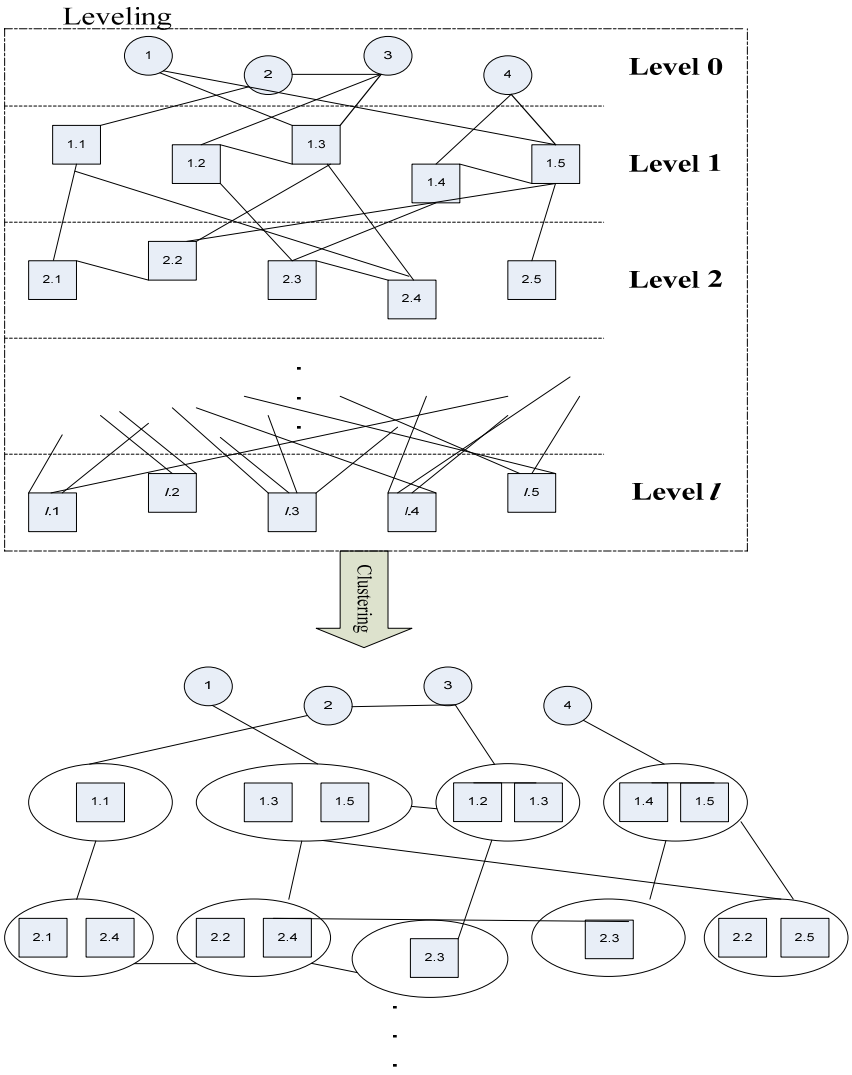


Figure 4: Leveling and clustering processes

Here, the clustered network is used to configure a tree (the Steiner tree) keeping the base functions and optimizing two objectives of minimal cost and maximal profit in the convergent product value adding process. Next, we formulate our adapted proposed Steiner tree model. In the proposed network, node i (function or sub-function i) have two costs:
c_{i1}: software cost,
c_{i2}: hardware cost.

Each arc is accompanied with a benefit $p_{ii'}$ which is obtained from nodes i and i'. With respect to the solution approach and using the Steiner tree in the proposed network and the NP-hardness of the problem, we used leveling and clustering processes to reduce the complexity of the problem. In clustering, it is not acceptable for any node to be included in more than one cluster in any level. To guarantee this, for each level l, a zero/one mathematical program is developed in order to properly appropriate nodes in clusters with the aim to minimize the total cost.

Next, we give the zero/one mathematical program and the purification procedure for each level.

The zero/one mathematical program for level *l*:

$$\min \quad T = \sum_{i \in n_l} \sum_{j \in m_{li}} (\alpha_{ij} c_{ij1} + \beta_{ij} c_{ij2}) z_{ij}$$

$$\sum_{j \in m_{li}} z_{ij} = 1 \qquad i = 1, ..., n_l$$

$$z_{ij} = \begin{cases} 1, & \text{if node } i \text{ is in cluster } j \\ 0, & \text{otherwise} \end{cases}$$

where, $\beta_{ij}, \alpha_{ij} \in [0,1], \forall i, j$,

m_{li} : set of indices of clusters at level l where node i is included,
n_l : set of indices of different nodes in level l,
c_{ij2} : hardware cost of node i in cluster j (c_{ij2}= c_{i2}, for all i and j),
c_{ij1}: software cost of node i in cluster j (c_{ij1}= c_{i1}, for all i and j),
α_{ij} : the software reduction cost coefficient of node i in cluster j,
β_{ij} : the hardware reduction cost coefficient of node i in cluster j.

Purifying benefits and costs at level *l*:

To determine the cost for each cluster at level l, we use

$$c_{jl} = \sum_{i \in n_l} (\alpha_{ij} c_{ij1} + \beta_{ij} c_{ij2}) - \sum_{\forall i, i'} p_{ii'},$$

where, $\beta_{ij}, \alpha_{ij} \in [0,1], \forall i, j$,

c_{jl} : cost of cluster j in level l
α_{ij} : the software reduction cost coefficient of node i in cluster j,
β_{ij} : the hardware reduction cost coefficient of node i in cluster j,
$p_{ii'}$: the benefit of an arc connecting node i in cluster j to node i' in cluster j, and c_{ij1}, c_{ij2}, and n_l are as defined above.

Also, to adjust the combined arc benefits in clusters, the following equation is used:

$$p_{jj'} = (1 + \gamma_{jj'}) \sum_{\forall i, i'} p_{ii'}$$

where, $\forall i, i', j, j'$,

$p_{jj'}$ is the adjusted arc benefit connecting cluster j to cluster j',

$p_{ii'}$ is the benefit of an arc connecting node i in cluster j to node i' in cluster j', and

$\gamma_{jj'}$ is the added value configured from nodes in clusters j and j'.

Algorithms 1 and 2 are transformed into Algorithm 3 using the aforementioned considerations. Also, each node should be in only one cluster at level l. The node having a minimal cost is chosen for the level l. Then, instead of using the zero/one mathematical program for level l, we can use step 3 of Algorithm 3. This leads a reduction of computations by avoiding the need for using the zero/one programs.

Algorithm 3: leveling and clustering in the network.

Step 0: Set the base functions at level 0. Let l=0.
Step 1: **While** sub-functions exist for processing **do**
Find sub-functions with a link to a function (or sub-function) at level l and place them at level l+1; Let l=l+1;
End while.
{l is the number of levels}
Step 2: Set each node at level 0 to be a cluster.
Step 3: **For** i=1 till l **do**
{Form clusters at level i}

Cluster all sub-functions at level i linked to a single parent at level i-1;
While $|n_i| > 0$ **do**
Select $k \in n_i$ such that $\alpha_{kp}c_{kp1} + \beta_{kp}c_{kp2} = \min_{j \in m_{ik}}\{\alpha_{kj}c_{kj1} + \beta_{kj}c_{kj2}\}$. Set $z_{kp} = 1$ and $z_{kj} = 0$, $\forall j \in m_{ik}$, $j \neq p$;
$n_i \leftarrow n_i - \{k\}$.
End while;
For j=1 till q_i **do** { q_i is the number of clusters in the level i}

$$c_{ji} = \sum_{i \in n_l}(\alpha_{ij}c_{ij1} + \beta_{ij}c_{ij2}) - \sum_{\forall i,i'} p_{ii'};$$

End for;
For j=1 till q_i **do**
For j'=1 till q_i **do**

$$p_{jj'} = (1 + \gamma_{jj'})\sum_{\forall i,i'} p_{ii'};$$

End for;
End for;
End for.
Step 4: **Stop**.

3. Mathematical formulation and the extended MOACS approach

Here, we first propose the mathematical model for the considered problem and then state the solution approach.

Mathematical formulation

We first recall the undirected Dantzig–Fulkerson–Johnson model for the convergent product Steiner tree problem (CPSTP) presented in (Costa et al., 2006). Let x_{ij} and y_i be binary variables associated with links $(i, j) \in E$ and clusters $i \in V$, respectively. Variable y_i is 1 if cluster i belongs to the solution, and is 0 otherwise. Similarly, variable x_{ij} is 1 if link (i, j) belongs to the solution, and is 0 otherwise. For $S \subseteq V$, define $E(S)$ as the set of links with both end nodes in S. Assume that terminals are the set N. The mathematical model can then be written as

Maximize $$\sum_{(i,j)\in E} p_{ij}.x_{ij}, \quad (1)$$

Minimize $$\sum_{i \in V} c_i.y_i, \quad (2)$$

Such that

$$\sum_{(i,j)\in E} x_{ij} = \sum_{i \in V} y_i - 1, \quad (3)$$

$$\sum_{(i,j)\in E(S)} x_{ij} \leq \sum_{i \in S - \{k\}} y_i, \quad \forall k \in S \subseteq V,\ \forall S : |S| \geq 2, \quad (4)$$

$$y_h = 1, \quad \forall h \in N, \quad (5)$$

$$x_{ij} \in \{0,1\}, \quad \forall i, j \in E, \quad (6)$$

$$y_i \in \{0,1\}, \quad \forall i \in V. \quad (7)$$

The objectives are to maximize the aggregated benefits and minimize the aggregated costs. Constraint (3) guarantees that the number of clusters in a solution is equal to the number of links minus one, and constraints (4) are the connectivity constraints. The number of constraints (4)

equals $2^{|V|}-|V|-1$. As a result, the number of variables and constraints are increased exponentially with respect to the number of clusters. Constraints (5) impose the terminal clusters to exist in the tree. Relations (6) and (7) show the variable types.

The extended MOPSO approach

As mentioned, the STP belong to a class of NP-hard problems. Thus, to solve medium to large-sized problems, an efficient bi-objective particle swarm optimization (BOPSO) algorithm is proposed.

Classic PSO

Particle swarm optimization (PSO) has roots in two main aspects. Perhaps more obvious are its ties to artificial life (A-life), in general, and to bird flocking, fish schooling, and swarming theory in particular. It is also related, however, to evolutionary computing, and has ties to both GA and evolutionary programming (Kennedy et al., 1995).

A swarm is composed of particles such as birds, fishes, bees, etc.. Each particle searches the area for food with its velocity and always remembers the best position found. This value is called *pbest*. In addition, each member of the swarm knows the best position found by its best informant or by the group globally. This value is called *gbest*. Therefore, there are three fundamental elements for the calculation of the next displacement of a particle:

1) According to its own velocity.
2) Towards its best performance.
3) The best performance of its best informant.

The way in which these three vectors are combined linearly via confidence coefficients is the basis of all versions of the "classic" PSO (Clerc, 2006). In the Proposed BOPSO the approach developed by Zhong (2008) is used.

The Proposed MOPSO

Because the discrete particle swarm optimization algorithm for the Steiner tree problem is more complicated than the standard PSO, we give its flowchart in Figure 5. There are several key components in the proposed algorithm, which are the preprocessing operations, the representation, the update operations, the improving strategies, updating velocity and position values.

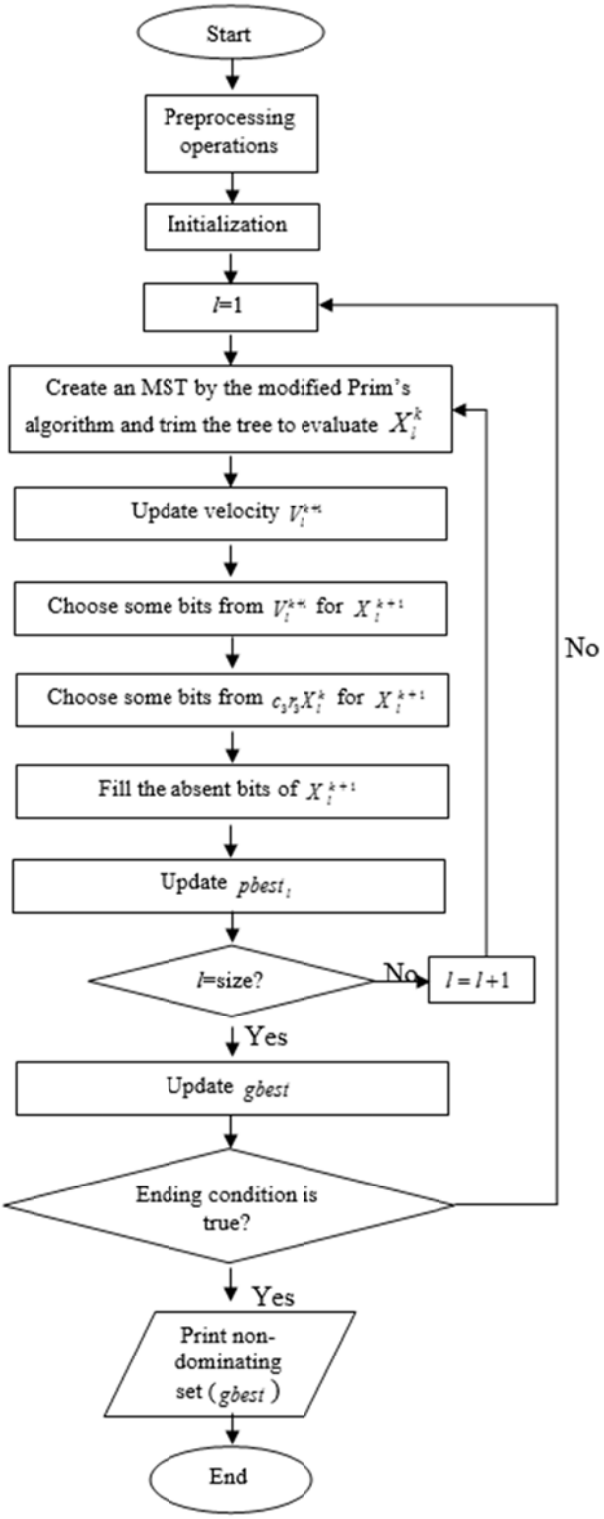


Figure 5: Flowchart of the multi objective PSO algorithm

The preprocessing operations

The solution of the STP is always a connected tree. However, because the graph G is not guaranteed to be a complete graph, a selected collection of nodes may not be adequate to represent a maximum spanning tree (MST). To overcome this obstacle, Floyd's algorithm

(Thomas et al., 1990) is applied to figure out the longest distance and the path between any two indirectly connected nodes in the graph. As a result, the graph is transformed to a complete graph.

The other preprocessing operation is finding, the longest distance from each node to all the terminal nodes. Then, the nodes are to be sorted according to the distance in increasing order. These points will be considered later in our latest modification, *Modification* 7, for the update operations.

The representation

As mention before, the key for STP is to find out the proper intermediate nodes and it is easy to generate a maximum spanning tree (MST) based on a binary string in a polynomial time. Consequently, the position of a particle is encoded as a binary string as $X_l^k = (x_{l1}, x_{l2}, ..., x_{\ln})$ in the proposed algorithm, where x_{li} is 0 or 1 and k denote the kth generation and the ith node in the graph, respectively. If $x_{li} = 1$, the ith node is selected as an intermediate node or it is a terminal node.

Remark 1: The bits standing for the terminal nodes are always set to 1.

The velocities are represented as

$$V = \begin{pmatrix} v_1^0, v_2^0, ..., v_n^0 \\ v_1^1, v_2^1, ..., v_n^1 \end{pmatrix}$$

such that $v_i^0 = \frac{\lambda}{c_i}$ and $v_i^1 = \frac{\gamma(c_i - 1)}{c_i}$,

where v_i^0 and v_i^1 are real numbers in the internal [0, 1] and denote the probabilities of the ith bit to be 0 or 1. Also, λ and γ are real numbers in the internal [0, 1]. According to *Modification* 5 below, the sum of v_i^0 and v_i^1 is not necessarily equal to 1.

The update equations

The particles are updated and evaluated repeatedly until the termination criterion is met. First, the parameters are initialized and every particle is given a random position and velocity. Then, the positions are saved as $pbest_l$, and evaluated by objective functions to select *gbest* or the non-dominating set. in the generations, a particle is updated by the following two equations:

$$V_l^{k+1} = \omega V_l^k + c_1 r_1 (pbest_l^k - X_l^k) + c_2 r_2 (gbest^k - X_l^k), \quad (8)$$

$$X_l^{k+1} = X_l^k + V_l^{k+1}, \quad (9)$$

where k denotes the kth generation. The first equation can be divided into three parts by the operator "+". The parameter ω, called the inertia weight (Shi and Eberhart, 1998, 2001), helps to control the influence of pre-velocity. The other two parts are the self-cognitive and the social-cognitive components. Both c_1 and c_2 are constant real positive numbers, which are used to direct the lth particle to $pbest_l$ and *gbest*. And r_1, r_2 are random real numbers belonging to [0, 1]. After the updating, the new positions will be evaluated again, and $pbest_l$ may be replaced and *gbest* may be updated, if necessary.

Since the standard PSO is designed for the continuous problems, the update equations (8) and (9) in the discrete particle swarm optimization algorithm for the Steiner tree problem have to be redefined for STP by the following 7 modifications.

Modification 1: The result of subtraction operator "–" between two positions (binary strings) X_1 and X_2 in (8) is defined as a velocity, in which v_i^b is set to 1 if the *j*th bit in X_1 is *b* while it is not in X_2. Otherwise, v_i^b is set to 0.

Modification 2: In equation (8), the result V of subtraction is multiplied by $c_j r_j$ (j = 1, 2), where c_j is a constant real number and r_j is a random real number belonging to [0, 1]. We do the same in our proposed algorithm. As a result, each element is multiplied by a unique real random number selected in [0, c_j], because the r_j are spawn randomly.

Remark 2: If v_i^b is greater than 1 after the multiplication, it will be set 1.

Remark 3: As to ωV, the result is that each element v_i^b in V is multiplied by the same ω.

Modification 3: The result of the operator "+" in equation (8) between two velocities is a new velocity, that is $V = V_1 + V_2$. The v_i^b in V is the greater one between v_{1i}^b and v_{2i}^b.

Following the three modifications above, the first equation is totally redefined for STP. An example is given as follows.

Assume $\omega = 0.5, r_1 = 2, r_2 = 2$ and $pbest_l^k = (1,1,0,0,1,1,1,0)$, $gbest^k = (1,1,1,0,1,0,1,0)$, $V_l^k = \begin{pmatrix} 0.4,0,0, & 0.2,0,0,1,0 \\ 0.6,0,0.2,0.8,0,0,0,1 \end{pmatrix}$.

So $V_{l1}^{k+1} = c_1 r_1 (pbest_l^k - X_l^k) = \begin{pmatrix} 0,0,0.8,0,0,0,0,0 \\ 0,0,0 \quad ,0,0,0,1,0 \end{pmatrix}$. Suppose that $c_1 r_1 = 0.8$ and 1.5 for each v_i^b ($v_i^b \neq 0$).

Let, $V_{l2}^{k+1} = c_2 r_2 (gbest^k - X_l^k) = \begin{pmatrix} 0,0,0,0,0,0.7,0 \quad ,0 \\ 0,0,0,0,0,0 \quad ,0.6,0 \end{pmatrix}$. Suppose that $c_2 r_2 = 0.7$ and 0.6 for each v_i^b ($v_l^b \neq 0$).

Then, $V_l^{k+1} = \omega V_l^k + V_{l1}^{k+1} + V_{l2}^{k+1} = \begin{pmatrix} 0.2,0,0.8,0.1,0,0.7,0.5,0 \\ 0.3,0,0.1,0.4,0,0 \quad ,1, 0.5 \end{pmatrix}$.

The second update equation is composed of four modifications.

Modification 4: The next position X_l^{k+1} is initialized to be an empty string.

Modification 5: Generate a random number $\alpha \in [0,1]$, and if v_i^b in V_l^{k+1} is greater than α, then the *i*th bit of X_l^{k+1} is set to *b*. If both v_i^0 and v_i^1 are greater than α, then the *i*th bit is set to 0 or 1 randomly. And, if both v_i^0 and v_i^1 are smaller than α, then the *i*th bit is set to "-", which stands for the absent bits. After these two steps, X_l^{k+1} is a string with several absent bit.

Modification 6: A new parameter c_3 is added to equation (9) to keep a balance between exploitation and exploration, that is, the second update equation is modified as:

$$X_l^{k+1} = V_l^{k+1} \otimes c_3 r_3 X_l^k .$$

For each absent bit in X_l^k, a random real number r_3 belonging to [0, 1] is spawned, and if $c_3 r_3 > \alpha$, then the corresponding bit in X_l^k is copied to X_l^{k+1}.

However, the number of $v_i^b = 0$ in V_l^{k+1} increases rapidly. So, if we choose as many as possible bits in X_l^k, then almost all elements in X_l^{k+1} will come from X_l^k, and the algorithm stagnates.

Modification 7: Because several bits are perhaps absent in X_l^{k+1}, a strategy to fill the blanks is adopted. For the *i*th absent bit, a random real number *t* belonging to [0, 1] is generated, if $t < rand_i / n$, where $rand_i$ is the rank of the *i*th node figured out in the preprocessing operations, and *n* is the number of nodes. Otherwise, the *i*th bit is set to 0. So, for each node, the farer to all the terminals it is, the more opportunity to be selected it will have.

Assume $\alpha = 0.5,\ c_3 = 2,$ and V_l^{k+1} is given as in the example after *Modification* 3. So, a possible case is $X_l^{k+1} = (-,-,0,-,-,0,0,1)$.

Assume $c_3 r_3 = 1, 0.5, 0.4, 0.7$ for the first, second, forth and the fifth bits, respectively. Then, the first, second and the fifth bits in X_l^k are copied to X_l^{k+1}, and so $X_l^{k+1} = (1,1,0,-,1,0,0,1)$.

Suppose $t < rand_4 / n$. Then $X_l^{k+1} = (1,1,0,0,1,0,0,1)$.

The modified Prim's algorithm and the trimming strategy

According to the seven modifications, X_l^{k+1} is a new *n* bit binary string. Because the solution of CPSTP is a maximum spanning tree, we need to design a method to convert a binary string to a corresponding MST, and then the benefits of the edges of MST and the cost of nodes of MST show the fitness of the particle. The Prim's algorithm (Zhong et al., 2008) is an effective algorithm to do such a work in a connected graph. However, because it is not known whether all the selected nodes in X_l^{k+1} are connected directly, we modify the prim's algorithm as follows.

Definition 1: A real edge is an edge connecting two nodes in the graph.

Definition 2: A virtual edge is the longest path connecting two nodes, which contain at least one intermediate node. This is figured out by Floyd's algorithm (Thomas et al., 1990) in the preprocessing operation. And, restoring a virtual edge means using the intermediate nodes and real edges to replace it.

Definition 3: *S* is the set containing nodes already involved in MST. And, ~*S* is the set whose members are candidate nodes for MST.

The modified Prim's algorithm is:

a. Choose a node randomly to be long to *S*, and put the other nodes in ~*S*.
b. Find out the longest real edge connecting one node in *S* and another in ~*S*. If succeeded, then move the selected node from ~*S* to *S* and repeat this step until ~*S* is empty. If no real edges exist between *S* and ~*S*, then go to c.
c. Find out the longest virtual edge connecting one node in *S* and another in ~*S*, and then move the selected node from ~*S* to *S* and record the virtual edge, and go to b. If ~*S* is empty, then go to d.
d. Restore the recorded virtual edges and terminate the modified Prim's algorithm.

Note that after termination, the MST containing all the selected nodes in X_l^{k+1} is generated. Due to the restoring operation, some additional nodes may also be involved in MST.

Updating Velocity and Position Values

In solving multi-objective problems by PSO, the most important item to be paid more attention is the selection procedure for *gbest* and *pbest* corresponding to each particle when the velocity and position of particles should be updated.

The analogy of PSO with evolutionary algorithms makes it evident that using a Pareto ranking scheme can be a straightforward way to extend the approach to handle multi-objective optimization problems. The historical record of best solutions found by a particle (i.e., an

individual) can be used to store non-dominating solutions generated in the past, being similar to the notion of elitism used in evolutionary multi-objective optimization. The use of global attraction mechanisms combined with a historical archive of previously found non-dominating vectors motivate convergence towards *Ptrue* (Coello Coello et al., 2007).

For each particle, there is a *pbest* archive that can save all the best positions found by the particle. Actually, these positions are non-dominating solutions and each particle has its own approximate Pareto solutions archive. When a new solution is obtained, it is compared with the ones in *pbest* archive based on the domination principle. If this new solution dominates any other in the *pbest* archive, then that archived solution is removed from the *pbest* archive and the new solution is placed in the archive. Moreover, there is a *gbest* archive for the group and the global best solutions are saved. This set of Pareto solutions are obtained using the fast non-dominated sorting procedure proposed by Deb et al. (2002). Updating the *gbest* archive is done like updating the *pbest* archive, that is, if a new solution dominates any other in the *gbest* archive, then that archived solution is removed from the *gbest* archive and the new solution is placed in the archive.

To maintain diversity in the search and escape from the local optima, each *gbest* has a chance to be selected. The steps of this selection procedure are presented below:

Step 1. Divide the number of the particles into the number of the *gbest archive*.

Step 2. Consider the remainder. If the remainder is equal to 0, then the number of times each member of *gbest* can be selected to update the velocity of the particles equals the sub-multiple value, and go to Step 4; otherwise, go to Step 3.

Step 3. Calculate the crowding distance proposed by Deb et al. (2002) for each member of *gbest*. Sort these obtained values in descending order and choose the first *k* members of *gbest* (as many as the remainder value). The number of times each member of *gbest*, except for these *k* members of *gbest*, can be selected to update the velocity of the particles is equal to the sub-multiple value. However, these *k* members of *gbest* have one more chance than other members to be selected.

Step 4. Select *gbest* and a *pbest* for each particle. To assign the *gbest* to a particle, the minimum Euclidean distance of that particle from the *gbests*, which still has a chance to be selected, is considered. It means that the nearest *gbest* to each particle, which still has a chance, is chosen. To assign a *pbest* to a particle, the *pbest* with the maximum Euclidean distance from the *gbest* assigned to that particle is chosen. Stop.

The following numerical example describes the mentioned steps clearly. Assume that a PSO algorithm has 90 particles and the number of members of *gbest* in the *gbest* archive is 7. If we divide 90 into 7, the sub-multiple value is equal to 12. So, each member of *gbest* can be selected 12 times. However, the remainder does not equal 0. It means that 6 particles do not have *gbest* values for updating velocity and position values. In this case, we calculate the crowding distance for each member of *gbest* and sort the resulting numbers in descending order. The first 6 members of *gbest* with large crowding distance values are considered. These members can be selected 13 times.

When MOPSO is iterated as many as a pre-specified value, the multi-objective optimization process is terminated and the set of solutions of the final *gbest* archive is reported as the Pareto/efficient solutions for the CPSTP problem.

4. An experimental study

Here, to illustrate the applicability and effectiveness of our proposed multiple optimization process, an experiment is worked out. Consider an undirected graph *G=(i, j)* with the cluster set $V = \{1,...,n\}$ and the link set $E = \{e = (i, j) : i, j \in V,\ i < j\}$, non-negative profits, p_e, associated with every link and non-negative costs, c_i, associated with the clusters. In this Steiner tree problem, the aim is to find the tree maximizing the revenue, i.e., the sum of the profits of the links in p_e spanned by the solution, and minimizing the sum of the costs of the clusters in the solution. On the one hand, we would like to have all links spanned by the solution avoiding the

loss of profit; but this can be too expensive in terms of the cost of the tree-structured network providing service to all clusters. Thus, there is a trade-off between the cost of the clusters being in the solution and the profit of the links by the solution.

The three dimensional matrix of functions, sub-functions, and attributes are shown in Table 1. Note that the tables related to all the three attributes are configured and their arithmetic means are shown as the final functions, sub-functions, and attributes comparison matrix. Our threshold value is considered to be 0.561 which is the mean of the data given in Table 1. Therefore, the thresholded matrix is shown in Table 2, and the corresponding network is configured as Figure 6.

Table 1: The three dimensional comparison matrix for all attributes.

All attributes	B_1	B_2	B_3	S_1	S_2	S_3	S_4	S_5	S_6	S_7
B_1	0	0.59	0.58	0.6	0.53	0.66	0.46	0.52	0.48	0.46
B_2	-	0	0.46	0.36	0.46	0.63	0.86	0.58	0.43	0.43
B_3	-	-	0	0.59	0.56	0.53	0.9	0.63	0.33	0.53
S_1	-	-	-	0	0.58	0.59	0.6	0.53	0.6	0.6
S_2	-	-	-	-	0	0.36	0.43	0.43	0.7	0.83
S_3	-	-	-	-	-	0	0.63	0.6	0.63	0.58
S_4	-	-	-	-	-	-	0	0.46	0.6	0.43
S_5	-	-	-	-	-	-	-	0	0.65	0.63
S_6	-	-	-	-	-	-	-	-	0	0.63
S_7	-	-	-	-	-	-	-	-	-	0

Table 2: The thresholded comparison matrix for all attributes.

All attributes	B_1	B_2	B_3	S_1	S_2	S_3	S_4	S_5	S_6	S_7
B_1	0	1	1	1	0	1	0	0	0	0
B_2	-	0	0	0	0	1	1	1	0	0
B_3	-	-	0	1	0	0	1	1	0	0
S_1	-	-	-	0	1	1	1	0	1	1
S_2	-	-	-	-	0	0	0	0	1	1
S_3	-	-	-	-	-	0	1	1	1	1
S_4	-	-	-	-	-	-	0	0	1	0
S_5	-	-	-	-	-	-	-	0	1	1
S_6	-	-	-	-	-	-	-	-	0	1
S_7	-	-	-	-	-	-	-	-	-	0

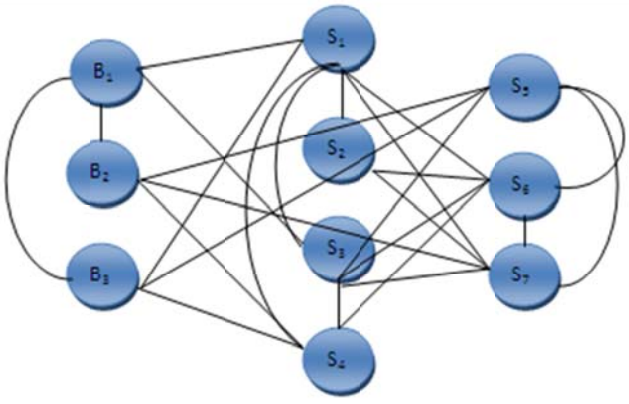


Figure 6: The configured thresholded network

Then, the leveling process (the zero th and the first steps of Algorithm 3) is performed and the leveled network is configured as Figure 7. The clustered network (the second and the third steps of Algorithm 3) is shown in Figure 8.

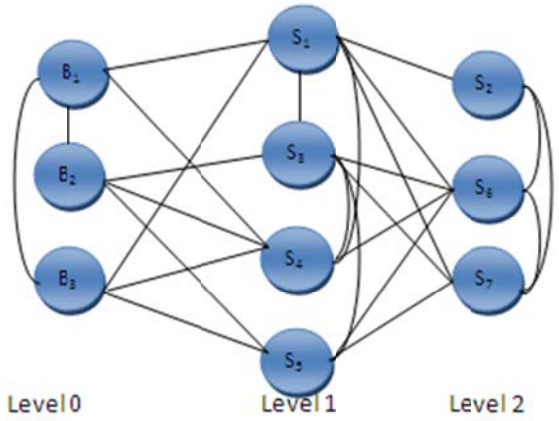


Figure 7: The configured leveled network

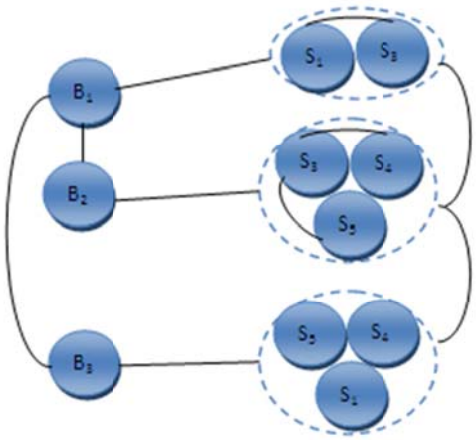


Figure 8: The configured clustered network (the second and the third steps of Algorithm 3)

The cost vertex, the benefit matrix and the matrices $\alpha = [\alpha_{ij}]$, $\beta = [\beta_{ij}]$ are:

C_1=(150, 210, 180, 20, 30, 30, 50, 10, 20, 20),
C_2=(300, 450, 600, 70, 80, 50, 40, 20, 20, 20).

$$\alpha = \begin{bmatrix} 0.65 & - & 0.7 \\ - & - & - \\ 1 & 0.8 & - \\ - & 0.6 & 0.9 \\ - & 0.7 & 0.65 \\ - & - & - \\ - & - & - \end{bmatrix} \quad \beta = \begin{bmatrix} 0.9 & - & 0.9 \\ - & - & - \\ 0.9 & 0.7 & - \\ - & 0.6 & 0.9 \\ - & 0.7 & 0.6 \\ - & - & - \\ - & - & - \end{bmatrix} \quad p = \begin{bmatrix} - & 1500 & 1300 & 1100 & - & 80 & - & - & - & - \\ - & - & - & - & - & 70 & 90 & 120 & - & - \\ - & - & - & 60 & - & - & 100 & 110 & - & - \\ - & - & - & - & 50 & 90 & 110 & - & 60 & 40 \\ - & - & - & - & - & - & - & - & 70 & 30 \\ - & - & - & - & - & - & 80 & 70 & 40 & 20 \\ - & - & - & - & - & - & - & - & 30 & - \\ - & - & - & - & - & - & - & - & 20 & 10 \\ - & - & - & - & - & - & - & - & - & 20 \\ - & - & - & - & - & - & - & - & - & - \end{bmatrix}$$

For level 1, using iteration 1 of the while loop in step 3 of Algorithm 3, we obtain:

$(\alpha_{41}c_{411} + \beta_{41}c_{412}) = 76$
$(\alpha_{43}c_{431} + \beta_{43}c_{432}) = 77$
$(\alpha_{61}c_{611} + \beta_{61}c_{612}) = 75$
$(\alpha_{62}c_{621} + \beta_{62}c_{622}) = 59$
$(\alpha_{72}c_{721} + \beta_{72}c_{722}) = 54$
$(\alpha_{73}c_{731} + \beta_{73}c_{732}) = 81$
$(\alpha_{82}c_{821} + \beta_{82}c_{822}) = 21$
$(\alpha_{83}c_{831} + \beta_{83}c_{832}) = 18.5$

Therefore, $z_{41} = 1, z_{62} = 1, z_{72} = 1$ and $z_{83} = 1$ with other variables equal to zero. The configured network up to level 1 is shown in Figure 9.

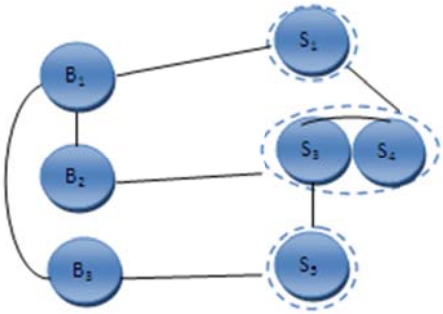


Figure 9. The configured clustered network for level 1

In Figure 10, the next iteration of Algorithm 3 for clustering is performed and the purified network is obtained, and the final network is obtained as Figure 11.

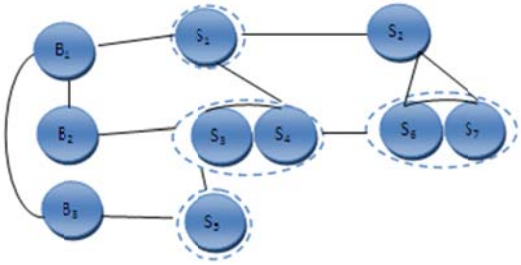


Figure 10. The configured clustered network

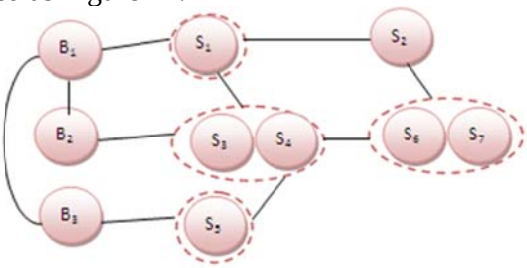


Figure 11. The configured clustered network

After purifying benefits and costs for level 2, the final cost and benefit matrices are formed as follows:

$$\tilde{p}=\begin{bmatrix} - & 1500 & 1300 & 1100 & - & - & - & - \\ - & - & - & - & - & 208 & - & - \\ - & - & - & - & - & - & 110 & - \\ - & - & - & - & 50 & 132 & - & - \\ - & - & - & - & - & - & - & 130 \\ - & - & - & - & - & - & 84 & 135 \\ - & - & - & - & - & - & - & - \\ - & - & - & - & - & - & - & - \end{bmatrix}$$

$$\tilde{c}=(450 \quad 660 \quad 780 \quad 90 \quad 110 \quad 33 \quad 90 \quad 41)$$

With respect to these matrices, the Steiner tree model is:

$$\max \quad X=1500x_{12}+1300x_{13}+100x_{14}+208x_{26}+110x_{37}+50x_{45}+132x_{46}+130x_{58}+84x_{67}+135x_{68}$$

$$\min \quad Y=450y_1+660y_2+780y_3+90y_4+110y_5+33y_6+90y_7+41y_8$$

s.t.

$$x_{12}+x_{13}+x_{14}+x_{26}+x_{37}+x_{45}+x_{46}+x_{58}+x_{67}+x_{68}=y_1+y_2+y_3+y_4+y_5+y_6+y_7+y_8-1$$

$$\sum_{(i,j)\in E(S)} x_{ij} \le \sum_{i\in S-\{k\}} y_i, \qquad \forall k\in S\subseteq V=\{1,2,3,4,5,5,6,7,8\},\ \forall S:|S|\ge 2$$

$$y_h=1, \qquad \forall h\in\{1,2,3\}$$

$$x_{ij}\in\{0,1\}, \qquad \forall i,j$$

$$y_i\in\{0,1\}, \qquad \forall i.$$

This model was solved using BOPSO in 100 iterations each having 20 particles in MATLAB 9 software environment. The Pareto solutions are obtained and shown in Table 3. Some of the Pareto solutions are shown in figures 13-16. Figure 12 shows the Pareto solutions in the two-dimensional space resulted from the two objective functions.

Table 3: The Pareto solutions resulted from the two objective functions.

The number of the Pareto solution	cost	benefit
1	2254	3515
2	2164	3405
3	2143	3385
4	2053	3275
5	1964	3143
6	1923	3008
7	1890	2800

In Figure 12, the X-axis is symmetry of benefit and the Y axis is cost.

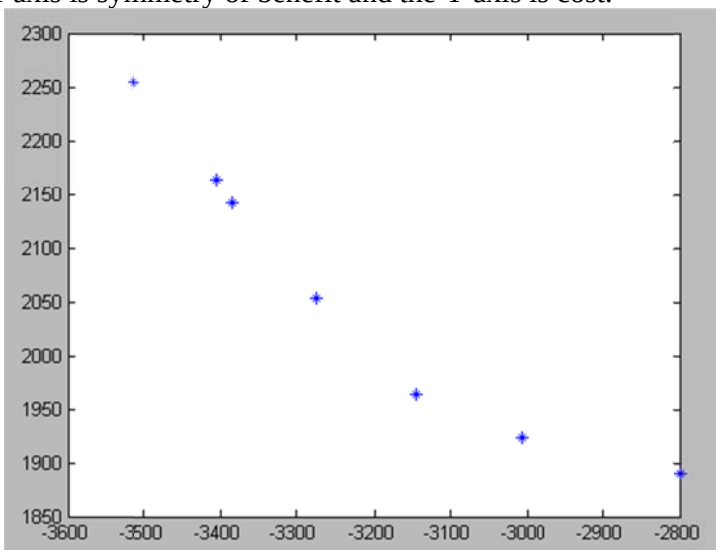


Figure 12: The Pareto solutions in two-dimensional space resulted from the two objective functions.

1) The first Pareto optimal solution is $X^* = 3515$ and $Y^* = 2254$, with the optimal network as shown in Figure 13.
2) The second Pareto optimal solution is $X^* = 3405$ and $Y^* = 2164$, with the optimal network as shown in Figure 14.

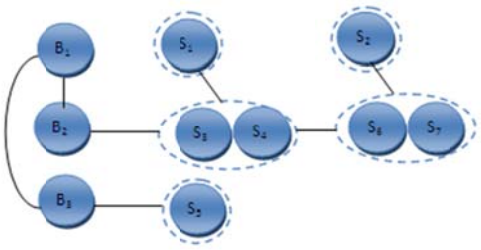


Figure 13: The first Pareto optimal solution

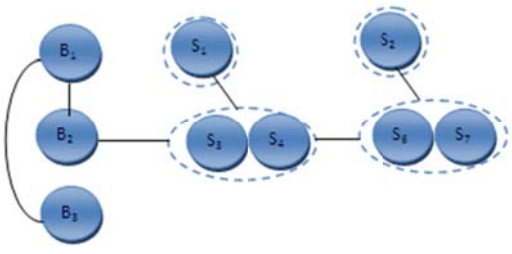


Figure 14: The second Pareto optimal solution

3) The fourth Pareto optimal solution is $X^* = 3275$ and $Y^* = 2053$, with the optimal network as shown in Figure 15.
4) The sixth Pareto optimal solution is $X^* = 3008$ and $Y^* = 1923$, with the optimal network as shown in Figure 16.

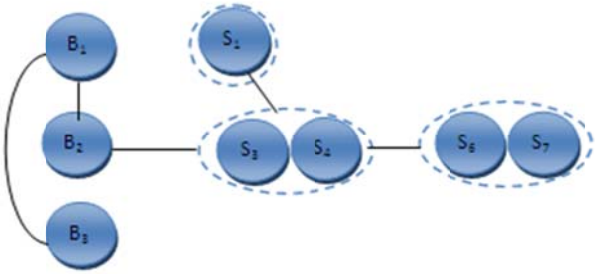


Figure 15: The fourth Pareto optimal solution

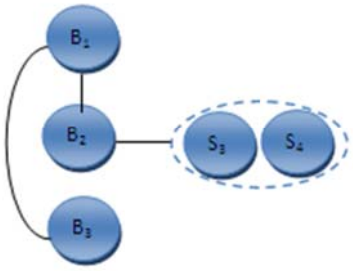


Figure 16: The sixth Pareto optimal solution

As shown in figures 13-16, the proposed method provides different products for producers and consumers having different benefits and costs. The numerical results imply the configuration of different products having various costs being based on customers' views obtained from the web based system. The products themselsves are the ones providing maximum benefits for the producers. The significant decision made in the proposed methodology is the trade-off between the cost and the benefit objectives based on the customers' views on adding features of products and producers' views on configuration of beneficial features.

5. Conclusions

We proposed a methodology to determine value adding functionalities for convergent products. A Steiner tree was modeled to handle the proposed network problem. In the network, each arc is assigned to be the link between two nodes. The aim was to find an optimal tree of functionalities in the network adding value to the product in the web environment. First, a purification process was performed in the product network to assign the links among bases and sub-functions. Then, numerical values as benefits and costs were determined for arcs and nodes, respectively, using leveling and clustering approaches. A bi-objective particle swarm optimization algorithm was developed to provide a solution framework. Several implications were discussed based on the obtained results.

Acknowledgements

The first two authors thank Mazandaran University of Science and Technology and the third author thanks Sharif University of Technology for supporting this work.

References

1. Beasley, J. E., l989. "An SST-based algorithm for the Steiner problem in graphs. *Networks*, 19(1), 1-16.
2. Clerc, M. 2006. *Particles Swarm Optimization*, ISTE Ltd.
3. Coello Coello, C.A., Lamont, G.B., Van Veldhuizen, D.A., 2007, *Evolutionary algorithms for solving multi-objective problems*, Springer Science+Business Media, LLC. 2nd edition.
4. Costa, A. M., Cordeau, J. F., Laporte, G., 2006. "Steiner tree problems with profits". *INFOR* 44(2), 99-115.
5. Deb, K., Pratap, A., Agarwal, S., Meyarivan, T. 2002. "A fast and elitist multiobjective genetic algorithm: NSGA-II", *IEEE Transactions on Evolutionary Computation* 6, 182-197.
6. Dreyfuss, S. E., & Wagner, R. A., 1971. "The Steiner problem in graphs". *Networks* 1(3), 195-207.
7. Duin, C. W., & Volgenant, A., 1989. "Reduction tests for the Steiner problem in graphs", *Networks* 19(5), 549-567.
8. Esbensen, H., 1995. "Computing near-optimal solutions to the Steiner problem in a graph using a genetic algorithm". *Networks*, 26(4), 173-185.

9. Fatta, G. Di, Presti, G. Lo, Re, G. Lo, 2003. "A parallel genetic algorithm for the Steiner problem in networks", 15t*h IASTED International Conference on Parallel and Distributed Computing and Systems (PDCS 2003)*, Marina del Rey, CA, USA, 569-573.
10. Hakimi, S. B., 1971. "Steiner's problem in graphs and its implications". *Networks*, 1, 113-133.
11. Hwang, F. K., & Richards, D. S., 1992. "Steiner tree problems". *Networks*, 22(1), 55-89.
12. Johnson, D. S., 1985. "The NP-completeness column: An ongoing guide". *Journal of Algorithms*, 6(3), 434-451.
13. Kapsalis, A., Rayward-Smith, V. J., Smith, G. D., 1993. "Solving the graphical Steiner tree problem using genetic algorithms". *Journal of the Operational Research Society*, 44(4), 397-406.
14. Kennedy, J., & Eberhart, R. 1995. "Particle Swarm optimization", *The 1995 IEEE international conference on neural networks*, 4, 1942-1948.
15. Koch, T., Martin, A., 1998. "Solving Steiner tree problems in graphs to optimalitys". *Networks*, 32(3), 207-232.
16. Kou, L., Markowsky, G., Berman, L., 1981. "A fast algorithm for Steiner trees". *Acta Informatica*, 15(2), 141-145.
17. Lawler, E. L., 1976. Combinatorial Optimbation Networks and Matroids, Holt, Rinehart and Winston, New York.
18. Martins, S. L., Pardalos, P., Resende, M.G., Ribeiro, C.C., 1999. "Greedy randomized adaptive search procedures for the Steiner problem in graphs". *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 43, 133-146.
19. Martins, S. L., Resende, M. G. C., Ribeiro, C.C., Pardalos, P.M., 2000. "A parallel GRASP for the Steiner tree problem in graphs using a hybrid local search strategy". *Journal of Global Optimization*, 17(1), 267-283.
20. Martins, S. L., Ribeiro, C. C., Souza, M. C., 1998. "A parallel GRASP for the Steiner problem in graphs". *Lecture Notes in Computer Science*, Springer-Verlag, 1457, 310-331.
21. Mehlhorn, K., 1988. "A faster approximation algorithm for the Steiner problem in graphs". *Information Processing Letters Archive*, 27(3), 125-128.
22. Rayward-Smith, V. J., Clare, A., 1986. "On finding Steiner vertices". *Networks*, 16(3), 283-294.
23. Ribeiro, C. C., Souza, M.C., 2000. "Tabu search for the Steiner problem in graphs". *Networks*, 36(2), 138-146.
24. Robins, G., Zelikovsky, A., 2000. "Improved Steiner tree approximation in graphs", *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*, SIAM, Philadelphia, ACM, New York, 770-779.
25. Robins, G., Zelikovsky, A., 2005. "Tighter bounds for graph Steiner tree approximation". *SIAM Journal on Discrete Mathematics* 19 (1), 122-134.
26. Shi, Y., Eberhart, R. C. 1998. "A modified particle swarm optimizer", *Proc. IEEE Int. Conf. Evol. Comput*., 69-73.
27. Shi, Y., & Eberhart, R. C. 2001. "Fuzzy adaptive particle swarm optimization", *Proc. IEEE Int. Congr. Evol. Comput*., 1, 101-106.
28. Takahashi, H., Matsuyama, A., 1980. "An approximate solution for the Steiner problem in graphs". *Mathematica Japonica*, 24(6), 573-577.
29. Thomas, H.C., Charles, E.L., Ronald, L. R., Clifford, S. 1990. *Introduction to Algorithms*, MIT press. USA.
30. Winter, P., & MacGregor Smith, J., 1992. "Path-distance heuristics for the Steiner problem in undirected networks". *Algorithmica*, 7(2 & 3), 309-327.
31. Zhong, W.L., Huang, J,. & Zhang, J. 2008. "A novel particle swarm optimization for the Steiner tree problem in graphs", *IEEE World Congress on Computational Intelligence.* Hong Kong.