Embedding complete multipartite graphs into Cartesian product of paths and cycles

R. Sundara Rajan\textsuperscript{a}, A. Arul Shantrinal\textsuperscript{a}, T.M. Rajalaxmi\textsuperscript{b}, Jianxi Fan\textsuperscript{c}, Weibei Fan\textsuperscript{d}

\textsuperscript{a}Department of Mathematics, Hindustan Institute of Technology and Science, Chennai, India, 603 103
\textsuperscript{b}Department of Mathematics, Sri Sivasubramaniya Nadar College of Engineering, Chennai, India, 603 110
\textsuperscript{c}School of Computer Science and Technology, Soochow University, Suzhou 215006, China
\textsuperscript{d}School of Computer Science & Technology, Nanjing University of Posts and Telecommunications, Jiangsu 210049, China

vprsundar@gmail.com, shandrinashan@gmail.com (Corresponding author), laxmi.raj18@gmail.com, jxfan@suda.edu.cn, fanweibei@163.com

Abstract

Graph embedding is a powerful method in parallel computing that maps a guest network \( G \) into a host network \( H \). The performance of an embedding can be evaluated by certain parameters, such as the dilation, the edge congestion and the wirelength. In this manuscript, we obtain the wirelength (exact and minimum) of embedding complete multi-partite graphs into Cartesian product of paths and/or cycles, which include \( n \)-cube, \( n \)-dimensional mesh (grid), \( n \)-dimensional cylinder and \( n \)-dimensional torus, etc., as the subfamilies.

Keywords: embedding, edge congestion, wirelength, complete multi-partite graphs, Cartesian product of graphs
Mathematics Subject Classification: 05C60, 05C85
DOI: 10.5614/ejgta.2021.9.2.21

1. Introduction and Preliminaries

Given two graphs \( G \) (guest) and \( H \) (host), an embedding from \( G \) to \( H \) is an injective mapping \( f : V(G) \to V(H) \) and associating a path \( P_f(e) \) in \( H \) for each edge \( e \) of \( G \). We, now define the edge congestion \( EC(G, H) \) and the wirelength \( WL(G, H) \) \cite{4} as follows:

\begin{itemize}
  \item \( EC(G, H) = \min_{f:G\to H} \max_{e=xy\in E(H)} EC_f(e) \)
  \item \( WL(G, H) = \min_{f:G\to H} \sum_{e=xy\in E(G)} \text{dist}_H(f(x), f(y)) = \min_{f:G\to H} \sum_{e=xy\in E(H)} EC_f(e) \)
\end{itemize}

Received: 27 September 2019, Revised: 19 June 2021, Accepted: 14 July 2021.
where \( \text{dist}_H(f(x), f(y)) \) is a distance (need not be a shortest distance) between \( f(x) \) and \( f(y) \) in \( H \) and \( EC_f(e) \) denote the number of edges \( e' \) of \( G \) such that \( e = xy \) is in the path \( P_f(e') \) (need not be a shortest path) between \( f(x) \) and \( f(y) \) in \( H \). Further, \( EC_f(S) = \sum_{e \in S} EC_f(e) \), where \( S \subseteq E(H) \).

For example, the edge congestion and the wirelength of an embedding \( f : C_3 \square C_3 \rightarrow P_9 \) is given in Fig. 1. It is easy to observe that, the above two parameters are different. But, for any embedding \( g \), the sum of the edge congestion (called edge congestion sum) and the wirelength are all equal. Mathematically, we have the following equality.

\[
\sum_{e=xy \in E(H)} EC_g(e) = WL_g(G, H).
\]

In this manuscript, we will use the edge congestion sum to estimate the wirelength. Further, if \( n \geq 1 \), then the set \( \{1, 2, \ldots, n\} \) will be denoted by \([n]\).

For a subgraph \( M \) of \( G \) of order \( n \),

- \( I_G(M) = \{uv \in E \mid u, v \in M\} \), \( I_G(k) = \max_{M \subseteq V(G), |M|=k} |I_G(M)| \)
- \( \theta_G(M) = \{uv \in E \mid u \in M, v \notin M\} \), \( \theta_G(k) = \min_{M \subseteq V(G), |M|=k} |\theta_G(M)| \)

The maximum subgraph problem (MSP) for a given \( k, k \in [n] \) is a problem of computing a subset \( M \) of \( V(G) \) such that \( |M| = k \) and \( |I_G(M)| = I_G(k) \). Further, the subsets \( M \) are called the optimal set \([17, 5, 19]\). Similarly, we define the minimum cut problem (MCP) for a given \( k, k \in [n] \) is a problem of computing a subset \( M \) of \( V(G) \) such that \( |M| = k \) and \( |\theta_G(M)| = \theta_G(k) \). For a regular graph, say \( r \), we have \( 2I_G(k) + \theta_G(k) = rk, k \in [n] \) \([5]\).

The following lemmas are efficient techniques to find the exact wirelength using MSP and MCP.

**Lemma 1.1.** \([27]\) Let \( f : G \rightarrow H \) be an embedding with \( |V(G)| = |V(H)| \). Let \( S \subseteq E(H) \) be such that \( E(H) \setminus S \) has exactly two subgraphs \( H_1 \) and \( H_2 \), and let \( G_1 = G[f^{-1}(V(H_1))] \) and \( G_2 = G[f^{-1}(V(H_2))] \). Furthermore, let \( S \) satisfy the following conditions:

1. For every \( uv \in E(G_1), i \in [2] \), the path \( P_f(uv) \) has no edges in \( S \).
2. For every \( uv \in E(G), u \in V(G_1), v \in V(G_2) \), the path \( P_f(uv) \) has exactly one edge in \( S \).
3. \( V(G_1) \) and \( V(G_2) \) are optimal sets.

Then \( EC_f(S) \) is minimum over all embeddings \( f : G \to H \) and

\[
EC_f(S) = \sum_{v \in V(G)} \deg_G(v) - 2|E(G_1)| = \sum_{v \in V(G_2)} \deg_G(v) - 2|E(G_2)|,
\]

where \( \deg_G(v) \) is the degree of a vertex \( v \) in \( G \).

Remark 1.1. For a regular graph \( G \), it is easy to check that, \( V(G_2) \) is optimal if and only if \( V(G_1) \) is optimal and vice-versa [25].

Lemma 1.2. [27] Let \( f : G \to H \). If \( \{P_1, \ldots, P_t\} \) is a partition of \( E(H) \), where each part \( P_i \) is an edge cut that satisfies the conditions of Lemma 1.1, then

\[
WL_f(G, H) = \sum_{i=1}^{t} EC_f(P_i).
\]

Moreover, \( WL(G, H) = WL_f(G, H) \).

The multipartite graph is one in all the foremost in style convertible and economical topological structures of interconnection networks. The multipartite has several wonderful options and its one in all the most effective topological structure of parallel processing and computing systems. In parallel computing, a large process is often decomposed into a collection of little sub processes which will execute in parallel with communications among these sub processes. Due to these communication relations among these sub processes the multipartite graph can be applied for avoiding conflicts in the network as well as multipartite networks helps to identify the errors occurring areas in easy way. A complete \( p \)-partite graph \( G = K_{n_1, \ldots, n_p} \) is a graph that contains \( p \) independent sets containing \( n_i, i \in [p] \), vertices, and all possible edges between vertices from different parts.

The Cartesian product technique is a very powerful technique for create a huger graph from given little graphs and it is very important technique for planning large-scale interconnection networks [45, 38]. Especially, the \( n \)-dimensional grid (cylinder and torus) structure of interconnection networks offer a really powerful communication pattern to execute a lot of algorithms in many parallel computing systems [45], which helps to arrange the interconnection network into sequence of sub processors (layers) in uniform distribution manner for transmits the data’s in faster way without delay in sending the data packets (messages). Mathematically, we now defined the Cartesian product of graphs as follows:

**Definition 1.1.** [20] The Cartesian product \( G \square H \) of (not necessarily connected) graphs \( G \) and \( H \) is the graph with the vertex set \( V(G) \times V(G) \), vertices \( (u, v) \) and \( (u', v') \) being adjacent if either \( u = u' \) and \( vv' \in E(H) \), or \( v = v' \) and \( uu' \in E(G) \). If \( G_1, G_2, \ldots, G_m \) are graphs of order \( n_1, n_2, \ldots, n_m \) respectively, then the Cartesian product of \( m \) factors \( G_1 \square G_2 \square \cdots \square G_m \) is denoted by \( \bigotimes_{i=1}^{m} G_i \).

**Remark 1.2.** The graph \( \bigotimes_{i=1}^{n} G_i \) is said to be an \( n \)-dimensional grid or torus or cylinder if all \( n \) factors are paths or cycles or any one of the factor is path and the remaining \( (n-1) \) factors are cycles, respectively.
The graph embedding problem has been well-studied by many authors with a different networks [1,5,6,10,11–45], and to our knowledge, almost all graphs considered as a host graph is a unique family (for example: path $P_n$, cycle $C_n$, grid $P_n \Box P_m$, cylinder $P_n \Box C_m$, torus $C_n \Box C_m$, hypercube $Q_r$ and so on). In this paper, we overcome this by taking Cartesian product of paths and/or cycles as a host graph. Moreover, we obtain the wirelength of embedding complete $2^p$-partite graphs $K_{2^{r_1}−p,2^{r_2}−p,...,2^{r_p}−p}$ into the Cartesian product of $n \geq 3$ factors of respective order $2^{r_i}$, $r_i \in \mathbb{N}$, where $r_1 \leq r_2 \leq \cdots \leq r_n$, $r_1 + r_2 + \cdots + r_n = r$ and each factor is a path or a cycle, $r \geq 3, 1 \leq p < r$.

**Lemma 1.3.** [30] If $G$ is a complete $p$-partite graph $K_{r,r,...,r}$ of order $pr$, $p, r \geq 2$, then

$$I_G(k) = \begin{cases} \frac{k(k-1)}{2}, & k \leq p-1, \\ \frac{q^2p(p-1)}{2}, & l = qp, 1 \leq q \leq r, \\ \frac{(q-1)^2p(p-1)}{2} + j(q-1)(p-1) + \frac{j(j-1)}{2}, & l = (q-1)p + j, 1 \leq j \leq p-1, \\ 2 \leq q \leq r. \end{cases}$$

**2. Main Results**

In this section we give an algorithm that computes the minimum wirelength of embedding complete $2^p$-partite graphs $K_{2^{r_1}−p,2^{r_2}−p,...,2^{r_p}−p}$ into Cartesian product of $n$ factors are paths or cycles or any one of the factor is path and the remaining $(n-1)$ factors are cycles.

We start with an auxiliary algorithm that accordingly labels the vertices of the complete $2^p$-partite graph $K_{2^{r_1}−p,2^{r_2}−p,...,2^{r_p}−p}$. We thus have $2^r$ vertices partitioned into $l = 2^p$ parts. Initially, all the vertices are unlabeled. Then label the first vertex in each partition (upto $l$) by increment of 1 using clockwise direction. Now, label the second vertex in the first partition as $l + 1$. Now, label the second vertex in the remaining each partition by increment of 1. Continue this process until all the $2^r$ vertices are labeled. The formal algorithm is given below as Algorithm 1.

**Algorithm 1:**

**Input:** $N = 2^r$ (Total number of elements) $\quad p \geq 1$, where $2^r-p$ represents the number of elements in the each partite

**Output:** Labeling of complete $2^p$-partite graph $K_{2^{r_1}−p,2^{r_2}−p,...,2^{r_p}−p}$

**Step 1.** Initialize $(z, x)$, $z$ represent the partite number and $y$ represent the vertex position of the individual partite considered in the loop

**Step 2.** Initialize $j = 1$

**Step 3.** Start the below loop for the first partite

**Step 4.** for ($x \leftarrow 1$ to $2^r-p$, increment $x$ by 1)

**Step 4.1.** for ($z \leftarrow 1$ to $2^p$, increment $z$ by 1)
Step 4.1.1. print \((z, x) = j\)

Step 4.1.2. Increment \(j\) value by 1

Step 5. Print the labeling of complete \(2^p\)-partite graphs

Step 6. Repeat Step 4 upto \(2^p\)-partite

Step 7. Repeat until \(2^{r-p}\) vertices are labeled in each partite.

Lemma 2.1. If \(r, n \geq 3\) and \(p \geq 1\), then Algorithm 1 labels the vertices of the complete \(2^p\)-partite graph \(K_{2^{r-p}, 2^{r-p}, \ldots, 2^{r-p}}\) with different integers from \([2^r]\).

Proof. The graph \(K_{2^{r-p}, 2^{r-p}, \ldots, 2^{r-p}}\) has \(2^r\) vertices partitioned into \(l = 2^p\) independent parts. Algorithm proceeds as described before. Specifically, Step 4.1.1 labels the first vertex of the first partition as 1, and continues the same process up to \(l\)th partition by increment of 1. The second vertex of the first partition is labeled with \(l+1\) and do the numbering in clockwise direction. Repeat the same process (Step 4) until we reach the label \(2^r\). Hence Algorithm 1 labels all the vertices of the complete \(2^p\)-partite graph uniquely from 1 to \(2^r\). This completes the proof of the lemma.

To illustrate Algorithm 1, consider the complete 4-partite graph \(K_{16, 16, 16, 16}\) as illustrated in Fig. 2. By Algorithm 1, we have \(N = 64\), \(r = 6\) and \(p = 2\). Initialize, \(j = 1\) for \(z = 1\) and \(x = 1\), \(z \in [16], x \in [16], (1, 1) = 1\) (i.e., \(x \in [2^{r-p}], z \in [2^p], (z, x) = j\) ) and hence label the first vertex of the first partite as 1. Next increment \(j\) by 1. For \(j = 2\), we have \(z = 2\) and \(x = 1\), \((1, 2) = 2\), Now label the first vertex of the second partite as 2 and so. Now go to Step 4, repeat the same process until we reach the label of the last (64th) vertex and the algorithm ends.

An implementation of Algorithm 1 in python and two of its outputs are listed in Annexure I.

We continue with an auxiliary algorithm that labels the Cartesian product of \(n \geq 3\) factors of respective order \(2^i, i \in [n]\), where \(r_1 + r_2 + \cdots + r_n = r, r_1 \leq r_2 \leq \cdots \leq r_n\) and each factor is a path or a cycle, \(r \geq 3, 1 \leq p < r\).

Algorithm 2:

Input: The dimension \(n \geq 3\) and the value of \(r_1, r_2, \ldots, r_n\)

Output: Labeling of Cartesian product of graphs \(\bigotimes_{i=1}^{n} G_i\), where \(G_i\)’s are either a path or a cycle
Step 1. Initialize $R$, $M$, $L$, and $(x, y)$, where $R$ represents the number of Cartesian product of $(n - 1)^{th}$ dimension graph, $M$ represents the number of two dimensional Cartesian product graph in the $(n - 1)^{th}$ dimension, $L = 2^{r}$, $r = r_1 + r_2 + \ldots + r_n$ (i.e., total number of vertices), and in $(x, y)$, $x$ represents the column and $y$ represents the row.

Step 2. Initialize variables $j = 1$, $P = 1$, $y = 1$, $Q = 2^{r_1}$, $K = 2^{r_2}$.


Step 4. for $(Z \leftarrow 1$ to $M$, increment $Z$ by 1)

// If $r_3 = r_n$, then $Z = 1$

Step 4.1. for $(x \leftarrow 1$ to $K$, increment $x$ by 1)

Step 4.2. $Q = 2^{r_1}$

Step 4.3. for $(y \leftarrow P$ to $Q$, increment $y$ by 1)

Step 4.3.1. print $(x, y) = j$

Step 4.3.2. if $\frac{j}{2^{r_1} \times 2^{r_2} \times M \times R} = 1$, then $P = y$

else

Step 4.3.3 if $(y = 2^{r_1})$, then $y = 0$ and $Q = P - 1$

Step 4.3.4 $j = j + 1$

Step 4.4. End for

Step 4.5. End for

Step 5. End for

Step 6. $R = R + 1$

Step 7. Repeat Step 4 for $2^n$ copies of $(n - 1)$ dimensional graph

Step 8. Print the labeling of the Cartesian product of paths and cycles.

Figure 3. The Cartesian product of path graphs $P_4 \square P_4 \square P_4$.

Lemma 2.2. If $r \geq 3$ and $i \in n$, then Algorithm 2 labels the Cartesian product of $n \geq 3$ factors of respective order $2^i$, where $r_1 \leq r_2 \leq \cdots \leq r_n$, $r_1 + r_2 + \cdots + r_n = r$, $1 \leq p < r$ and each factor is a path or a cycle.
Proof. The graph has $2^r$ vertices and dimension $n \geq 3$. Algorithm 2 proceeds as described before. We initialize the first two $a_1$ and $a_2$ parameter as the two dimensional graph. The $a_3$ parameter takes input which produces $a_3$ copies of the base dimension $(a_1 \times a_2)$. Similarly, the $a_n$ parameter takes input which produces $a_n$ copies of the base dimension $(a_1 \times a_2 \cdots \times a_{n-1})$. First label the $(n-1)$ dimensional graph. Then, update the row position and start the labeling in the second copy of $(n-1)$ dimensional graph. Repeat the same process (Step 4) until we reach the label $2^r$. Hence Algorithm 2 labels the all the vertices of the Cartesian product of paths and cycles uniquely from 1 to $2^r$. This completes the proof of the lemma.

To illustrate Algorithm 2, we consider the graph $P_4 \Box P_4 \Box P_4$ as illustrated in Fig. 3. In Algorithm 2, we have $Q = 4$ and $K = 4$. Initialize, $R = 1$, $j = 1$, $P = 1$ for $Z = 1$, $x \in [4]$, $y \in [4]$, $(x, y) = (1, 1) = 1$ and hence label the first vertex of the 2-dimensional grid as 1. Then, go to Step 4.3.2, the condition $\frac{2^r \times 2^2 \times M \times R}{4 \times 4 \times 1 \times 1} \neq 1$ fails. Next go to Step 4.3.3, the condition $y = 2^r \Rightarrow 1 \neq 4$ fails. Now increment $j$ by 1. Go to Step 4.3, we have $j = 2$, $x = 1$ and $y = 2$, $(1, 2) = 2$. Now, label the first vertex of the second row as 2 and so, for $j = 5$ we have $x = 1$ and $y = 5$, the condition $y \nless Q = 5 \nless 4$ fails. Thus, go to Step 4.1, we have $x = 2$ and $y = 1$, $(2, 1) = 5$ hence label the second vertex of the first row as 5. Continue this process until we reach all vertices of the two dimensional grid. Next go to Step 4.3.2, we have $(4, 4) = 16$, so the condition $\frac{2^r \times 2^2 \times M \times R}{4 \times 4 \times 1 \times 1} = 1$ satisfied. So, take the increment of $R$ by 1. Now, $P = y$ (i.e., $P = 4$), so $(x, y) = (1, 4) = 17$, label the first vertex of the last row as 17. Go to Step 4.3.2, the condition fails. Now come to the else part, we have $4 = 4 \ (y = 2^r_1)$. So, $y = 0$, $Q = P - 1 = 4 - 1 = 3$. Now increment $j$ by 1. Do the same process until we reach the label of the last $4^{th}$ vertex.

The Python program and the corresponding implementation of Algorithm 2 are given in Annexure II.

We, now ready to prove the main result.

Theorem 2.1. Let $G$ be the complete $2^p$-partite graphs $K_{2^{r_1} \cdot 2^{r_2} \cdot \ldots \cdot 2^{r_p}}$ and $H$ be the Cartesian product of $n \geq 3$ factors of respective order $2^{r_1}$, $i \in [n]$, where $r_1 \leq r_2 \leq \cdots \leq r_n$, $r_1 + r_2 + \cdots + r_n = r$ and each factor is a path or a cycle, $r \geq 3, 1 \leq p < r$. Let $s \geq 0$ factors of $H$ are paths and the remaining $(n - s)$ factors are cycles. Then the embedding $f$ of $G$ into $H$ given by $f(x) = x$ with minimum wirelength and is given by,

$$WL(G, H) = \frac{2^{2r-p}(2^p-1)}{6} \left( 2^{r_1+2r_2+\cdots+2r_s} \left( \frac{1}{2^{r_1}} + \frac{1}{2^{r_2}} + \cdots + \frac{1}{2^{r_s}} \right) \right) + 2^{2r-p-3}(2^p-1)(2^{r_{s+1}}+2^{r_{s+2}}+\cdots+2^{r_n}).$$

Proof. Label the vertices of $G$ using Lemma 2.1 from 1 to $2^r$. Since the graph $H$ contains an $n$-dimensional grid and label the vertices of $n$-dimensional grid using Lemma 2.2 from 1 to $2^r$. For illustration, see Figures 2, 3, 4, 5 and 6. Let us assume that, the label represent each of the vertex, which is allocated by the above algorithms. Let $f: G \rightarrow H$ be an embedding and let $f(v) = v$
for all \( v \in V(G) \) and for \( uv \in E(G) \), let \( P_f(uv) \) be a path (shortest) between \( f(u) \) and \( f(v) \) in \( H \). Now, we have the following 3 cases.

**Case 1.** \( s = n \)

It is clear that, the graph \( H \) becomes an \( n \)-dimensional grid \( P_{2r_1} \Box P_{2r_2} \Box \cdots \Box P_{2r_n} \). For all \( i, j, 1 \leq i \leq n \) and \( 1 \leq j \leq 2^{r_i} - 1 \), let \( S^j_i \) be the edge cut of \( P_{2r_1} \Box P_{2r_2} \Box \cdots \Box P_{2r_n} \) consisting of the edges between the \( j^{th} \) and \((j+1)^{th} \) copies of \( P_{2r_1} \Box P_{2r_2} \Box \cdots \Box P_{2r_{i-1}} \Box P_{2r_{i+1}} \Box \cdots \Box P_{2r_n} \). Then \( \{ S^j_i : 1 \leq i \leq n \) and \( 1 \leq j \leq 2^{r_i} - 1 \} \) is an edge-partition of \( P_{2r_1} \Box P_{2r_2} \Box \cdots \Box P_{2r_n} \).

For all \( i, j, 1 \leq i \leq n \) and \( 1 \leq j \leq 2^{r_i} - 1 \), \( G^j_i \) and \( \overline{G}^j_i \) be the subgraph of the inverse images of \( V(H^j_i) \) and \( V(\overline{H}^j_i) \) under \( f \) respectively. By Lemma 2.1, \( \deg_G(v) = 2^{r_i} - p(2^p - 1)j \), for all \( v \in V(G^j_i) \) and hence \( I_G(2^{r_i} - p(2^p - 1)j) = 2^{2r_i - 2r_j - p(2^p - 1)}(2^{2r_i - 2r_j - p(2^p - 1)}j) / 2 \). By Case 2 of Lemma 1.3, \( E(G^j_i) \) is the maximum subgraph on \( |V(G^j_i)| = 2^{2r_i - 2r_j}j \) vertices in \( G \). Thus the edge cut \( S^j_i \) fulfil all the conditions of Lemma 1.1. Therefore

\[
EC_f(S^j_i) = 2^{2r_i - 2r_j - p(2^p - 1)}j - \left( \frac{2^{2r_i - 2r_j - p(2^p - 1)}j^2}{2} \right)
\]

is minimum for \( 1 \leq i \leq n \) and \( 1 \leq j \leq 2^{r_i} - 1 \).

Then by Lemma 1.2,

\[
WL(G, H) = \sum_{i=1}^{n} \sum_{j=1}^{2^{r_i} - 1} EC_f(S^j_i)
\]

\[
= \sum_{i=1}^{n} \sum_{j=1}^{2^{r_i} - 1} 2^{2r_i - r_j - p(2^p - 1)}(2^{r_i} - j)j
\]

\[
= \frac{2^{2r_i - p(2^p - 1)} - 1}{6} \left( 2^{2r_1} + 2^{2r_2} + \cdots + 2^{2r_n} - \left( \frac{1}{2^{r_1}} + \frac{1}{2^{r_2}} + \cdots + \frac{1}{2^{r_n}} \right) \right).
\]

**Case 2.** \( s = 0 \)

It is clear that, the graph \( H \) becomes an \( n \)-dimensional torus \( C_{2r_1} \Box C_{2r_2} \Box \cdots \Box C_{2r_n} \). For all \( i, j, 1 \leq i \leq n \) and \( 1 \leq j \leq 2^{r_i} - 1 \), let \( T^j_i \) be the edge cut of \( C_{2r_1} \Box C_{2r_2} \Box \cdots \Box C_{2r_n} \) consisting of the edges between the \( (2^{r_i} - i + j)^{th} \) \& \( (2^{r_i} - i + j + 1)^{th} \) and \( (2^{r_i} - i + j)^{th} \) \& \( (2^{r_i} - i + j + 1)^{th} \) copies of \( C_{2r_1} \Box C_{2r_2} \Box \cdots \Box C_{2r_{i-1}} \Box C_{2r_{i+1}} \Box \cdots \Box C_{2r_n} \). Then \( \{ T^j_i : 1 \leq i \leq n \) and \( 1 \leq j \leq 2^{r_i} - 1 \} \) is an edge-partition of \( C_{2r_1} \Box C_{2r_2} \Box \cdots \Box C_{2r_n} \).

For all \( i, j, 1 \leq i \leq n \) and \( 1 \leq j \leq 2^{r_i} - 1 \), \( E(H) \setminus T^j_i \) has two components \( H^j_i \) and \( \overline{H}^j_i \), where \( |V(H^j_i)| = |V(\overline{H}^j_i)| = 2^{r_i - 1} \). Let \( G^j_i \) and \( \overline{G}^j_i \) be the induced subgraph of the inverse images of \( V(H^j_i) \) and \( V(\overline{H}^j_i) \) under \( f \) respectively. By Lemma 2.1, \( \deg_G(v) = 2^{2r_i - 2r_j - 2p(2^p - 1)} \), for all \( v \in V(G^j_i) \) and hence \( I_G(2^{2r_i - 2r_j - 2p(2^p - 1)}) = 2^{2r_i - 2r_j - 2p(2^p - 1)} / 2 \). By Case 2 of Lemma 1.3, \( E(G^j_i) \) is the maximum subgraph on \( |V(G^j_i)| = 2^{r_i - 1} \) vertices in \( G \). Thus the edge cut \( T^j_i \) fulfil all the conditions
Embedding complete multipartite graphs into Cartesian products  

| R. Sundara Rajan et al. |

of Lemma 1.1. Therefore

\[
EC_f(T^j_i) = 2^{r-p}(2^p - 1)2^{r-1} - 2 \left(\frac{2^{2r-2p-2}2^p(2^p - 1)}{2}\right)
\]

\[
= 2^{2r-p-2}(2^p - 1)
\]

is minimum for \(1 \leq i \leq n\) and \(1 \leq j \leq 2^{ri-1}\).

Then by Lemma 1.2,

\[
WL(G, H) = \sum_{i=1}^{n} \sum_{j=1}^{2^{ri-1}} EC_f(T^j_i)
\]

\[
= \sum_{i=1}^{n} \sum_{j=1}^{2^{ri-1}} 2^{2r-p-2}(2^p - 1)
\]

\[
= 2^{2r-p-3}(2^p - 1)(2^{r_1} + 2^{r_1} + \cdots + 2^{r_n}).
\]

**Case 3.** \(n > s > 0\)

In this case, we describe \(H\) as the Cartesian product of \(n \geq 3\) factors of respective order \(2^{r_i}, i \in [n]\) where \(i \in [n]\), where \(r_1 \leq r_2 \leq \cdots \leq r_n, r_1 + r_2 + \cdots + r_n = r\) and each factor is a path or a cycle, \(r \geq 3, 1 \leq p < r\).

Let \(s > 0\) factors of \(H\) are paths and the other \((n - s)\) factors are cycles, then obtained by using the associativity of the Cartesian product and writing that,

\[
P_{2^{r_1}} \square \cdots \square P_{2^{r_s}} \square C_{2^{r_{s+1}}} \square \cdots \square C_{2^{r_n}} = (P_{2^{r_1}} \square \cdots \square P_{2^{r_s}}) \square (C_{2^{r_{s+1}}} \square \cdots \square C_{2^{r_n}})
\]

Let \(S^l_i, 1 \leq i \leq s, 1 \leq l \leq 2^{ri} - 1\) and \(T^m_i, s + 1 \leq i \leq n, 1 \leq m \leq 2^{ri-1}\) be the edge cuts of paths and cycles of \(H\) respectively.

By similar arguments in Case 1 and Case 2, we get

\[
EC_f(S^l_i) = 2^{2r-2r_i-p}(2^p - 1)(2^{ri} - l_l)
\]

is minimum for \(1 \leq i \leq s\) and \(1 \leq l \leq 2^{ri} - 1\) and

\[
EC_f(T^m_i) = 2^{2r-p-2}(2^p - 1)
\]

is minimum for \(s + 1 \leq i \leq n\) and \(1 \leq m \leq 2^{ri-1}\).
Then by Lemma 1.2,

\[ WL(G, H) = \sum_{i=1}^{s} \sum_{l=1}^{2^{r_i}-1} EC_f(S_i^l) + \sum_{i=s+1}^{n} \sum_{m=1}^{2^{r_i}-1} EC_f(T_i^m) \]

\[ = \sum_{i=1}^{s} \sum_{l=1}^{2^{r_i}-1} 2^{2^{r_i}-2r_i-p}(2^p - 1)(2^{r_i} - l)l + \sum_{i=s+1}^{n} \sum_{m=1}^{2^{r_i}-1} 2^{2^{r_i}-p-2}(2^p - 1)(2^{2r_i} - 1) \]

\[ = \frac{1}{6} \sum_{i=1}^{s} 2^{2^{r_i}-r_i-p}(2^p - 1)(2^{r_i} - 1)(2^{r_i} + 1) + \sum_{i=s+1}^{n} 2^{2^{r_i}+r_i-p-3}(2^p - 1) \]

\[ = \frac{2^{2r-p}(2^p - 1)}{6} \left[ \left( 2^{r_1} + 2^{r_2} + \cdots + 2^{r_s} \right) - \left( \frac{1}{2^{r_1}} + \frac{1}{2^{r_2}} + \cdots + \frac{1}{2^{r_s}} \right) \right] + 2^{2r-p-3}(2^p - 1)(2^{r_{s+1}} + 2^{r_{s+2}} + \cdots + 2^{r_n}) \]

\[ \square \]

**Corollary 2.1.** If \( G_1 \) is a path on \( 2^{r_1} \) vertices and \( G_i \) is a cycle on \( 2^{r_i} \) vertices, \( 2 \leq i \leq n \), then the host graph becomes an \( n \)-dimensional cylinder \( P_{2^{r_1}} \square C_{2^{r_2}} \square \cdots \square C_{2^{r_n}} \) and the wirelength of an embedding \( f \) from \( G \) into \( H \) is given by

\[ WL(G, H) = \frac{2^{2r-p}(2^p - 1)}{6} \left[ 2^{r_1} - \frac{1}{2^{r_1}} \right] + 2^{2r-p-3}(2^p - 1)(2^{r_{s+1}} + 2^{r_{s+2}} + \cdots + 2^{r_n}) \]

### 3. Conclusion and Future Work

In this manuscript, we found the wirelength (exact and minimum) of an embedding complete multi-partite graphs into Cartesian product of paths and/or cycles. Computing the dilation [4] and the edge congestion of embedding complete multi-partite graphs into Cartesian product and other product of graphs are under investigation.

**Acknowledgement**

The work of R. Sundara Rajan was partially supported by Project no. ECR/2016/1993, Science and Engineering Research Board (SERB), Department of Science and Technology (DST), Government of India. Further, we thank Prof. Gregory Gutin and Prof. Stefanie Gerke, Royal Holloway, University of London, TW20 0EX, UK; Prof. Indra Rajasingh, School of Advanced Sciences, Vellore Institute of Technology, Chennai, India and Dr. N. Parthiban, Department of Computer Science and Engineering, SRM Institute of Science and Technology, Chennai, India, for their fruitful suggestions. Further, the authors would like to thank the anonymous referees for their comments and suggestions. These comments and suggestions were very helpful for improving the quality of this paper.
Embedding complete multipartite graphs into Cartesian products  |  R. Sundara Rajan et al.

References


Embedding complete multipartite graphs into Cartesian products  |  R. Sundara Rajan et al.


Annexure I

Python program for labeling of the guest graph

def printline(num, boxes, boxesinrow):
    i = 0;
    string = ‘ ’
    for i in range(0, boxesinrow):
        for j in range(0, 4):
            string = string + ”{:<3d} ”.format(num + boxes * j) + ‘ ‘
        string = string + ‘ ‘
        num = num + 1
    print(string)

def printbox(num, boxes, elements_per_box, boxes_in_row):
    value=num
    temp = elements_per_box // 4
    while temp > 0:
        printline(num, boxes, boxes_in_row)
        temp = temp - 1
        num = num + boxes * 4
    print(‘\n’)
    return (value+boxes_in_row)

def printpattern(numl):
    boxes = len(numl)
    elements_Per_box = numl[0]
    num = 1
    temp = boxes
    while temp > 0:
        if temp >= 4:
            num = printbox(num, boxes, elements_Per_box, 4)
        else:
            num = printbox(num, boxes, elements_Per_box, temp)
        temp = temp - 4
    printpattern([32,32,..,32])
Implementation of the above Python program

Output 1:

```
I                  II                  III                 IV
1  9  17  25       2  10  18  26       3  11  19  27       4  12  20  28
33 41  49  57      34  42  50  58      35  43  51  59      36  44  52  60
65  73  81  89     66  74  82  90      67  75  83  91      68  76  84  92
97 105 113 121     98 106 114 122     99 107 115 123     100 108 116 124
129 137 145 153    130 138 146 154    131 139 147 155    132 140 148 156
161 169 177 185    162 170 178 186    163 171 179 187    164 172 180 188
193 201 209 217    194 202 210 218    195 203 211 219    196 204 212 220
225 233 241 249    226 234 242 250    227 235 243 251    228 236 244 252

V                  VI                  VII                 VIII
5  13  21  29      6  14  22  30      7  15  23  31      8  16  24  32
37 45  53  61      38  46  54  62      39  47  55  63      40  48  56  64
69  77  85  93     70  78  86  94      71  79  87  95      72  80  88  96
101 109 117 125    102 110 118 126    103 111 119 127    104 112 120 128
133 141 149 157    134 142 150 158    135 143 151 159    136 144 152 160
165 173 181 189    166 174 182 190    167 175 183 191    168 176 184 192
197 205 213 221    198 206 214 222    199 207 215 223    200 208 216 224
229 237 245 253    230 238 246 254    231 239 247 255    232 240 248 256
```

Figure 3: Complete 8-partite graph $K_{32,32,...,32}$
**Output 2:**

<table>
<thead>
<tr>
<th></th>
<th>I</th>
<th>II</th>
<th>III</th>
<th>IV</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>65</td>
<td>33</td>
<td>34</td>
<td>35</td>
<td>36</td>
</tr>
<tr>
<td>97</td>
<td>66</td>
<td>67</td>
<td>68</td>
<td>69</td>
</tr>
<tr>
<td>129</td>
<td></td>
<td>130</td>
<td>131</td>
<td>132</td>
</tr>
<tr>
<td>193</td>
<td></td>
<td>194</td>
<td>195</td>
<td>196</td>
</tr>
<tr>
<td>225</td>
<td></td>
<td>226</td>
<td>227</td>
<td>228</td>
</tr>
<tr>
<td>V</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>37</td>
<td>36</td>
<td>37</td>
<td>38</td>
<td>39</td>
</tr>
<tr>
<td>69</td>
<td>68</td>
<td>69</td>
<td>70</td>
<td>71</td>
</tr>
<tr>
<td>101</td>
<td></td>
<td>102</td>
<td>103</td>
<td>104</td>
</tr>
<tr>
<td>133</td>
<td></td>
<td>134</td>
<td>135</td>
<td>136</td>
</tr>
<tr>
<td>191</td>
<td></td>
<td>192</td>
<td>193</td>
<td>194</td>
</tr>
<tr>
<td>229</td>
<td></td>
<td>230</td>
<td>231</td>
<td>232</td>
</tr>
<tr>
<td>X</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>9</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>41</td>
<td>40</td>
<td>41</td>
<td>42</td>
<td>43</td>
</tr>
<tr>
<td>73</td>
<td>72</td>
<td>73</td>
<td>74</td>
<td>75</td>
</tr>
<tr>
<td>105</td>
<td></td>
<td>106</td>
<td>107</td>
<td>108</td>
</tr>
<tr>
<td>137</td>
<td></td>
<td>138</td>
<td>139</td>
<td>140</td>
</tr>
<tr>
<td>201</td>
<td></td>
<td>202</td>
<td>203</td>
<td>204</td>
</tr>
<tr>
<td>233</td>
<td></td>
<td>234</td>
<td>235</td>
<td>236</td>
</tr>
<tr>
<td>XIII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>13</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>45</td>
<td>44</td>
<td>45</td>
<td>46</td>
<td>47</td>
</tr>
<tr>
<td>77</td>
<td>76</td>
<td>77</td>
<td>78</td>
<td>79</td>
</tr>
<tr>
<td>109</td>
<td></td>
<td>110</td>
<td>111</td>
<td>112</td>
</tr>
<tr>
<td>141</td>
<td></td>
<td>142</td>
<td>143</td>
<td>144</td>
</tr>
<tr>
<td>205</td>
<td></td>
<td>206</td>
<td>207</td>
<td>208</td>
</tr>
<tr>
<td>237</td>
<td></td>
<td>238</td>
<td>239</td>
<td>240</td>
</tr>
<tr>
<td>XVII</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>17</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>49</td>
<td>48</td>
<td>49</td>
<td>50</td>
<td>51</td>
</tr>
<tr>
<td>81</td>
<td>80</td>
<td>81</td>
<td>82</td>
<td>83</td>
</tr>
<tr>
<td>113</td>
<td></td>
<td>114</td>
<td>115</td>
<td>116</td>
</tr>
<tr>
<td>145</td>
<td></td>
<td>146</td>
<td>147</td>
<td>148</td>
</tr>
<tr>
<td>209</td>
<td></td>
<td>210</td>
<td>211</td>
<td>212</td>
</tr>
<tr>
<td>241</td>
<td></td>
<td>242</td>
<td>243</td>
<td>244</td>
</tr>
<tr>
<td>XXI</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>21</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>53</td>
<td>52</td>
<td>53</td>
<td>54</td>
<td>55</td>
</tr>
<tr>
<td>85</td>
<td>84</td>
<td>85</td>
<td>86</td>
<td>87</td>
</tr>
<tr>
<td>117</td>
<td></td>
<td>118</td>
<td>119</td>
<td>120</td>
</tr>
<tr>
<td>149</td>
<td></td>
<td>150</td>
<td>151</td>
<td>152</td>
</tr>
<tr>
<td>213</td>
<td></td>
<td>214</td>
<td>215</td>
<td>216</td>
</tr>
<tr>
<td>245</td>
<td></td>
<td>246</td>
<td>247</td>
<td>248</td>
</tr>
<tr>
<td>XXV</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>25</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>57</td>
<td>56</td>
<td>57</td>
<td>58</td>
<td>59</td>
</tr>
<tr>
<td>89</td>
<td>88</td>
<td>89</td>
<td>90</td>
<td>91</td>
</tr>
<tr>
<td>121</td>
<td></td>
<td>122</td>
<td>123</td>
<td>124</td>
</tr>
<tr>
<td>153</td>
<td></td>
<td>154</td>
<td>155</td>
<td>156</td>
</tr>
<tr>
<td>217</td>
<td></td>
<td>218</td>
<td>219</td>
<td>220</td>
</tr>
<tr>
<td>249</td>
<td></td>
<td>250</td>
<td>251</td>
<td>252</td>
</tr>
<tr>
<td>XXIX</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>29</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>61</td>
<td>60</td>
<td>61</td>
<td>62</td>
<td>63</td>
</tr>
<tr>
<td>93</td>
<td>92</td>
<td>93</td>
<td>94</td>
<td>95</td>
</tr>
<tr>
<td>125</td>
<td></td>
<td>126</td>
<td>127</td>
<td>128</td>
</tr>
<tr>
<td>157</td>
<td></td>
<td>158</td>
<td>159</td>
<td>160</td>
</tr>
<tr>
<td>221</td>
<td></td>
<td>222</td>
<td>223</td>
<td>224</td>
</tr>
<tr>
<td>253</td>
<td></td>
<td>254</td>
<td>255</td>
<td>256</td>
</tr>
</tbody>
</table>

**Figure 4:** Complete 32-partite graph $K_{8,8,...,8}$
Annexure II

Python program for labeling of the host graph

```
n=0
def disp_3nr(numl, n, irange):
    for i in irange:
        string = str(i+n)
        for j in range(1, numl[1]):
            string+=str(i+n+numl[0]*j)
        for k in range(1, numl[2]):
            string=string+' ' + str(i+ n + numl[0] * j+ k*numl[0]*numl[1])
        print(string)
def disp_n(numl):
    base=numl[0]*numl[1]*numl[2]
    para=len(numl)
    order=numl[3:-]
    global n
    n = -base
    tnuml=numl[:3]
    loopri(order, tnum1, base)
def rotate(irange):
    temp=irange[0]
    for i in range(0, len(irange)-1):
        irange[i] = irange[i+1]
    irange[len(irange)-1]=temp
    return(irange)
def loopri(order, tnuml, base):
    irange = list(range(1, tnuml[0]+1))
    if len(order) == 1:
        for i in range(0, order[0]):
            global n
            print(i+1)
            n = n + base
            disp_3nr(tnuml, n, irange)
            irange=rotate(irange)
            print(' ')
    else:
        for i in range(0, order[-1]):
            print(i)
            loopri(order[:-1], tnuml, base, irange)
            irange=rotate(irange)
def loopr(order, tnuml, base, irange):
    if len(order) == 1:
        for i in range(0, order[0]):
            global n
            n = n + base
            disp_3nr(tnuml, n, irange)
            print(' ')
    else:
        for i in range(0, order[-1]):
            print(i)
            loopr(order[:-1], tnuml, base, irange)
def disp_n([4,8,16])
```

523 www.ejgta.org
Implementation of the above Python program

Output 1:

Figure 5: 3-dimensional grid $P_4 \Box P_8 \Box P_{16}$
Output 2:

Figure 6: 4-dimensional cylinder $C_4 \square P_4 \square P_4 \square P_8$