TensorFlow Job Placement Optimization as a Submodular Function Maximization

Amanda Carbonari, Nodir Kodirov, Maximilian Golub

May 1, 2017

Abstract

Distributed TensorFlow job execution enables large models to be trained and served in production. However, distributed job placement is one of the challenges associated with distributed job execution. In this project, we model the job placement problem as a submodular function maximization and apply standard greedy submodular algorithm to minimize job completion time.

We implemented the algorithm and compared it to the state-of-the-art heuristic greedy algorithm. Our experiments with three TensorFlow image recognition models with up to 311 nodes and physical topology with up to 60 devices shows that our submodular algorithm finds node placement with up to 2x slow job completion time. More importantly, we report submodular model's power to capture additional runtime constraints, such as available device memory and inter-device data transmission cost, which are impossible or prohibitively expensive to capture with heuristic model.

1 Introduction

A prime advantage of running TensorFlow job on a distributed setting is to leverage device heterogeneity. Different devices such as CPUs, GPUs, FPGAs, and ASICS like the Google TPU, have different capabilities making some more fit to run certain operation than others. For example, memory intensive operations like *concatenation* are better run on CPU as it has large amount of RAM available compared to GPU. Similarly, GPUs are much faster at running parallel computation, making them better fit for matrix operations like *convolution*.

Another advantage of distributed execution is scalability. It enables users to run larger models and make computation complete faster by adding more devices. However, the relationship between an additional device and job completion time is not a simple one. In fact, optimal job to device placement is NP-hard [Wik17]. Therefore, in practice, heuristic algorithms are used to decide job placement. For example, the greedy heuristic algorithm described in the seminal TensorFlow work traverses job nodes in their topological order and places nodes on the device where it completes fastest [AAB+15].

A TensorFlow job is described by a directed graph, which is composed of a set of nodes and job placement involves assigning nodes on a given set of devices while satisfying a predefined objective. Objectives can vary based operator's preference, but in practice common job placement objectives are handful, such as (1) minimum job completion time, (2) memory bound job execution, (3) load-balanced job execution, and (4) minimum inter-device data transmission.

In this project we solve objective (1) by formulating job placement problem as a submodular function maximization problem. We run placement algorithm and assume job topology is known in advance. This is the same assumption made in the greedy heuristic algorithm described above. In fact, in addition to job topology, the seminal work states availability node operations types (such as *convolution*), physical network of devices (such as CPU, GPU, TPU and connectivity between them), and time cost of each operation (such as *convolution* takes 2ns in GPU, 5ns in CPU, etc.) as input to the placement algorithm [AAB+15]. Reducing the problem definition into a submodular model subject to a matroid constraint allows us to utilize the submodular function maximization algorithms covered in the class, such as standard greedy, and reason about optimality of our node placement algorithm.

In the next sections we describe submodularity formulation with a proof, evaluate our algorithm against the greedy heuristics, and describe modeling power of submodular formalization to capture additional placement constraints.

2 Submodular Model

To describe the Tensorflow job placement problem as a submodular function, we first outline some key characterizations that would lead to a close to optimal placement. We found that there are three characteristics of node placement that are necessary:

1. Parallelism: placing a parallel node on a different device
2. Minimized cost: placing a node on a device that will minimize the run time
3. Device diversity: spread out the nodes over all devices

Our model must capture these three things for it to be effective. We could have tried to use submodular minimization, but it would have been difficult to model the problem such that it would place all nodes. This is one of the main constraints of the node placement problem. Technically, a partial placement is not a valid placement. Therefore, our model must always force the algorithm to choose a full placement, we achieve this using maximization of benefit, instead of minimization of cost as we explain in section 2.2.

2.1 Assumptions

To simplify the node placement problem, we make a few assumptions about the devices and setup our model handles. First, we assume that there is no memory constraint, each device has an infinite amount of memory. Secondly, we don't model the amount of data flowing through the system and the memory it requires at each device. Thirdly, we do not consider the cost of transferring data, we assume that the data transfer cost is 0.

We made these assumptions to simplify our model so we could establish a submodularity proof before attempting to make the model more robust. All of these assumptions could be added into our model by modifying the appropriate equations described below, we leave this discussion to our future work (section 5).

2.2 Definition

We solve node placement in Tensorflow using a submodular function subject to a matroid constraint. We define our matroid as $M = (E, \mathcal{I})$, where E is an m, n matrix, where m is the number of devices and n is the number of nodes, comprised of different $E_{i,j}$ rows, where i represents a particular device and j represents a particular node. i and j can be formally defined as $i \in 1..n$ and $j \in 1..m$. $E_{i,j}$ represents the placement of node j on device i. Therefore, E represents the full placement. For example,

$$E = \left\{ \begin{array}{l} E_{2,1} = [0 \quad c_{2,1} \quad 0] \\ E_{2,2} = [0 \quad c_{2,2} \quad 0] \\ E_{1,3} = [c_{1,3} \quad 0 \quad 0] \\ E_{3,4} = [0 \quad 0 \quad c_{3,4}] \\ E_{1,5} = [c_{1,5} \quad 0 \quad 0] \end{array} \right\}$$

$\mathcal{I}$ is the independent set of E where $\mathcal{I} = \{S \subseteq E : \quad |S \cap E^i| \leq 1 \quad \forall i\}$, which means $\mathcal{I}$ only contains one placement per node j. For example, $\mathcal{I}$ cannot contain the set $\{E_{11}, E_{21}\}$ because this would place node 1 on both device 1 and device 2, which is not a valid placement.

We define the submodular function as follows. Let $f(S) = 2^E \rightarrow \mathbb{R}$ and $f(S) = \sum_i f^i(S)$, which means $f(S)$ is the sum of the device specific f functions, where S is a set of $E_{i,j}$ values which comprise a placement. $f^i(S) = g(\sum_{j \in J_i} b_{i,j})$, where J_i is the set of all nodes assigned to device i and $b_{i,j}$ is the benefit of placing node j on device i. $g(x)$ can be any function which describes a diminishing return, in our initial model we use $g(x) = \sqrt{x}$. $b_{i,j}$ is defined in relation to the cost of putting node j on device i. Having a device specific f function allows us to capture the third characteristic listed above, device diversity. We can further model this using a specific $g(x)$ function.

The key to this model and maintaining the submodularity property is in the definition of $b_{i,j}$, the benefit function. The benefit must reflect the remaining two characteristics: (1) parallelism and (2) minimum completion time. To reflect both requirements, some assumptions are made about the benefit function:

- The topology of S is available and represented as a set of dependencies.
- The "already placed" nodes for this particular placement must be known.

For example, if the maximization has already placed node j' on device i', $b_{i,j}$ must know of this placement. This assumption is not a constraint on the algorithm because the placement information is already available in S. The benefit function requires this knowledge to determine parallelism. The more difficult part of $b_{i,j}$ is how to express the node completion time as something that can be maximized but will not invalidate the submodular properties (i.e., concave function). Our solution is as follows:

$$freeRatio_{i,j} = 2 - \left(\frac{c_{i,j} + placed\_nodes_i}{i_{cap}}\right)$$

The function $freeRatio_{i,j}$ captures the amount of "free" time left for that device. i_{cap} is the total "time" capacity the device i has, this can be an arbitrarily large number, such as `INTMAX`. This does constrain our solution, but if a sufficiently large i_{cap} is used, then it should not cause an inaccurate result. Due to the use of $g(x) = \sqrt{x}$, the resulting $freeRatio_{i,j}$ cannot be less than one, or else the square root will produce a number larger than the input. Therefore, we use 2 instead of 1. $placed\_nodes_i$ is the total cost of all the nodes currently placed on i, more succinctly: $placed\_nodes_i = \sum_{j \in J_i} c_{i,j}$ where J_i is the set of all nodes placed on i.

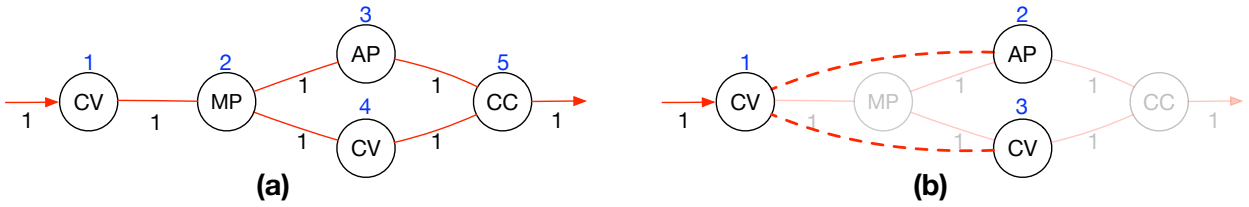

Figure 1: Simple job with 5 nodes. Node IDs are above vertices on blue (1, 2, 3, 4, 5) and data size exchanged between nodes are below edges on black. (a) shows full graph with dependency $D_S = [\emptyset, \{1\}, \{2\}, \{2\}, \{3, 4\}]$ where $S = \{E_1, E_2, E_3, E_4, E_5\}$, and (b) partial graph where dependency is $D_S = [\emptyset, \{1\}, \{1\}]$ where $S = \{E_1, E_3, E_4\}$.

Before explaining the parallelism factor, first we will describe what the dependency lists look like with a full and partial topology. Then we will describe how the parallelism factor is calculated using this dependency list (D_S). In the case of a full topology, the index in the list corresponds to $j - 1$ and the set at that index is the set of all nodes j depends on. For example, $D_S = [\emptyset, \{1\}, \{2\}, \{2\}, \{3, 4\}]$ would be the full topology dependency set for the example figure shown in Figure 1a. If S only contains nodes 1, 3, and 4 as shown in Figure 1b, then the dependency would be $D_S = [\emptyset, \{1\}, \{1\}]$. Parallelism can be detected because two nodes (2 and 3 in the partial topology, 3 and 4 in the full topology) have the same dependency. $pf_{i,j}$ depends on what nodes have already been placed on that device, in the example given above, if the S already contains nodes 1 and 3 placed on devices 1 and 2 respectively, then $pf_{1,4} = 2$, $pf_{2,4} = 1$ and $pf_{3,4} = 2$. The parallelism factor is the number of parallel nodes **if** the node j is being placed on a device which does not contain a parallel node. To determine if the node is parallel, it first looks to see if it shares any dependencies with other nodes (i.e., it does not depend on that node and both nodes depend on the same thing). Once it has those "parallel candidates", it determines if i is the same device that "parallelism candidate" is placed on. If i is not the same, then that is a "parallel node". It does this check for all "parallelism candidates". At the end, $pf_{i,j} = |\text{parallel nodes}|$. Therefore, $pf_{i,j}$ can be defined as follows:

$$\begin{aligned} PC_j &= \{j' : \text{if } j \neq j' \text{ and } D_S[j] \neq D_S[j'] \quad \forall j' \in 1..n\} \\ P_S^j &= \{j' : \text{if } j'_{device} \neq i \quad \forall j' \in PC_j\} \\ pf_{i,j} &= |P_S^j| \end{aligned}$$

Combining the percent idle time and parallelism, we get:

$$\begin{aligned} b_{i,j} &= (percentIdle_{i,j}) \cdot pf_{i,j} \\ &= (2 - \left(\frac{c_{i,j} + placed\_nodes_i}{i_{cap}}\right)) \cdot pf_{i,j} \end{aligned}$$

We use submodular maximization with the greedy algorithm to determine a placement. At the end of the greedy algorithm, it will output a placement S. From this placement S we can calculate a node completion time (JCT) as follows: sum all values, if node j and node j' are parallel, only add the $max(j, j')$. So it will be the sum of all sequential nodes and the most expensive parallel nodes.

2.2.1 Important Equations

To summarize, the important equations for our submodular model along with their variable definitions are outlined below.

Matroid.

$$
\begin{aligned}
M &= (E, \mathcal{I}) \\
E &= \left\{ \begin{array}{c|c|c|c} E_{1,1} & E_{2,1} & \cdots & E_{m,1} \\ E_{1,2} & E_{2,2} & \cdots & E_{3,2} \\ \vdots & \vdots & \cdots & \vdots \\ E_{1,n} & E_{2,n} & \cdots & E_{m,n} \end{array} \right\} \\
\mathcal{I} &= \{S \subseteq E : \quad |S \cap E^i| \leq 1 \quad \forall i\}
\end{aligned} \tag{1}
$$

Submodular Functions.

$$
\begin{aligned}
f(S) &= \sum_i f^i(S) \\
f^i(S) &= g(\sum_{j \in J_i} b_{i,j}) \\
&\text{where } J_i \text{ the set of all nodes assigned to device } i
\end{aligned} \tag{2}
$$

Helper Functions.

$$
\begin{aligned}
g(x) &= \sqrt{x} \\
b_{i,j} &= \left(2 - \left(\frac{c_{i,j} + placed\_nodes_i}{i_{cap}}\right)\right) \cdot pf_{i,j} \\
placed\_nodes_i &= \sum_{j \in J_i} c_{i,j} \\
&\text{where } J_i \text{ the set of all nodes assigned to device } i \\
pf_{i,j} &= |P_S^j| \\
P_S^j &= \{j' : \text{if } j'_{device} \neq i \quad \forall j' \in PC_j\} \\
PC_j &= \{j' : \text{if } j \neq j' \text{ and } D_S[j] \neq D_S[j'] \quad \forall j' \in 1..n\}
\end{aligned} \tag{3}
$$

- $c_{i,j}$ represents time cost of the node j on device i.
- i_{cap} the capacity of device i (constant).
- $pf_{ij} = 0$ if $S = \{\emptyset\}$

Job Completion Time.

$$
JCT(E) = \sum_{j=1}^{n} \begin{cases} \max\left(\forall j' \in P_E^j \quad E_{j'}\right), & \text{if } j \text{ is a parallel node and hasn't been counted} \\ 0, & \text{if } j \text{ is a parallel node and has been counted} \\ S_j, & \text{otherwise} \end{cases} \tag{4}
$$

Where P_E^j is P_S^j (defined above) where $E = S$.

In appendix, we walk-through placement of the 5 node job described in figure 1a on a physical network with 3 devices shown in figure below. This is intended to provide intuition behind functions and variables, and make our proof of submodularity in section 3 more descriptive.

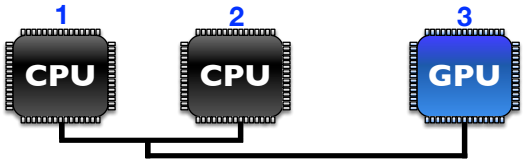

Figure 2: Two CPU cores and a GPU core available for node placement.

2.3 Improvements

The current model for parallelism causes the greedy algorithm to avoid placing on devices which exploit parallelism. It is obvious that this is not the desired behavior. In order to circumvent this behavior, we move the parallelism factor out of the benefit function and into the f function. We modify the equations given in Section 2.2 as follows:

$$\begin{aligned} b_{i,j} &= 2 - \frac{c_{i,j} - placed\_node_i}{i_{cap}} \\ f(S) &= \Big(\sum_i f^i(S)\Big) \cdot pf_{ij} \end{aligned} \tag{5}$$

Now the parallelism factor is captured after summing all the benefits over all devices. This should preserve submodularity but also not affect the f values later on. We do compare its JCT to the original model and Tensorflow's current algorithm in section 4.

3 Proof of Submodularity

To prove $f(S)$ is submodular, we first need to prove that $f^i(S)$ is submodular, as the sum of submodular functions is still a submodular function. In the case of our improved model, multiplying by $pf_{i,j}$ does not affect the submodularity because $pf_{ij} \geq 1$ [Von10, KG12]. We prove submodularity for our original model because this is the model we describe in Section 2.

Claim: $f(S)$ is a submodular function.

Proof. Let S be an arbitrary set where $S \in \mathcal{I}$ from the matroid $M = (E, \mathcal{I})$. The function $f(S) \to \mathbb{R}$ is computed using the following equations and assumptions:

$$\begin{aligned} f(S) &= \sum_i f^i(S) \\ f^i(S) &= g\Big(\sum_{j \in J_i} b_{i,j}\Big) \\ g(x) &= \sqrt{x} \\ b_{i,j} &= \Big(2 - \frac{c_{i,j} + placed\_job_i}{i_{cap}}\Big) \cdot pf_{i,j} \end{aligned} \tag{1}$$

where J_i represents jobs already placed on machine i, and c_ij, $placed\_job_i$, i_{cap}, and $pf_{i,j}$ are domain specific values which have the following constraints:

$$\begin{aligned} c_{i,j} &\geq 0 \\ placed\_job_i &\geq 0 \\ c_{i,j} + placed\_job_i &\leq i_{cap} \quad \forall i, j \\ pf_{i,j} &\geq 1 \end{aligned} \tag{2}$$

Therefore, $b_{i,j}$ must be greater than or equal to one, based on the minimum and maximum possible values for its variables, outlined below.

$$\begin{aligned} \textbf{min: } b_{i,j} &= \Big(2 - \frac{i_{cap}}{i_{cap}}\Big) \cdot 1 = 1 \\ \textbf{max: } b_{i,j} &= \Big(2 - \frac{0+0}{i_{cap}}\Big) \cdot w = 2 \cdot w \end{aligned} \tag{3}$$

For $f(S)$ to be submodular, $f^i(S)$ must be submodular. For $f^i(S)$ to be submodular, it must satisfy the following inequality:

$$f^i(S \cup \{E_{ia}\}) - f^i(S) \geq f^i(S \cup \{E_{ia}, E_{ib}\}) - f^i(S \cup \{E_{ib}\}) \tag{4}$$

Where a and b are arbitrary elements in the matroid not in S, formally $a, b \notin S$. Intuitively, this inequality captures *diminishing return of element* a, where a yields equal to or greater than value when added to a small set, compared to the case when it is added to a larger set. In other words, the marginal value of element a added to the set S (small set) is at least equal to the marginal value of a added to the set S with an added element b (larger set: $\{S \cup \{b\}\}$).

To simplify the proof, we assign values for a and b so they match the notation we are using: $a = E_{ia}$ and $b = E_{ib}$. To simplify the substitution, we define b_{iS} to represent the sum benefit of placing nodes on machine i. This can also be defined as the sum of all the b_{ip} values, where p job is placed on machine i and $E_{ip} \in S$, but $p \neq a, b$. More concisely: $b_{iS} = \sum_{p \in P_S} b_{ip}$ where $P_S = \{z : E_{iz} \in S \text{ and } z \neq a, b \quad \forall z\}$. Now substituting these values into equation 4, we get:

$$\sqrt{b_{iS} + b_{ia}} - \sqrt{b_{iS}} \geq \sqrt{b_{iS} + b_{ib} + b_{ia}} - \sqrt{b_{iS} + b_{ib}} \tag{5}$$

Now we complete the proof by contradiction. Let's assume that equation 5 is not correct, basically we assume the opposite is true:

$$\sqrt{b_{iS} + b_{ia}} - \sqrt{b_{iS}} < \sqrt{b_{iS} + b_{ib} + b_{ia}} - \sqrt{b_{iS} + b_{ib}} \tag{6}$$

With algebraic transformation we get following:

$$\begin{aligned}
\sqrt{b_{iS} + b_{ia}} + \sqrt{b_{iS} + b_{ib}} &< \sqrt{b_{iS} + b_{ib} + b_{ia}} + \sqrt{b_{iS}} \\
\sqrt{b_{iS} + b_{ia}} + \sqrt{b_{iS} + b_{ib}} &< \sqrt{b_{iS} + b_{ib} + b_{ia}} + \sqrt{b_{iS}} \qquad \text{| square both sides} \\
(\sqrt{b_{iS} + b_{ia}} + \sqrt{b_{iS} + b_{ib}})^2 &< (\sqrt{b_{iS} + b_{ib} + b_{ia}} + \sqrt{b_{iS}})^2 \\
(b_{iS} + b_{ia}) + 2\sqrt{(b_{iS} + b_{ia})(b_{iS} + b_{ib})} + (b_{iS} + b_{ib}) &< (b_{iS} + b_{ib} + b_{ia}) + 2\sqrt{(b_{iS} + b_{ib} + b_{ia}) \cdot b_{iS}} + b_{iS} \\
(\cancel{b}_{iS} + \cancel{b}_{ia}) + 2\sqrt{(b_{iS} + b_{ia})(b_{iS} + b_{ib})} + (\cancel{b}_{iS} + \cancel{b}_{ib}) &< (\cancel{b}_{iS} + \cancel{b}_{ib} + \cancel{b}_{ia}) + 2\sqrt{(b_{iS} + b_{ib} + b_{ia}) \cdot b_{iS}} + \cancel{b}_{iS} \\
2\sqrt{(b_{iS} + b_{ia})(b_{iS} + b_{ib})} &< 2\sqrt{(b_{iS} + b_{ib} + b_{ia}) \cdot b_{iS}} \qquad \text{| div to 2, apply square} \\
(b_{iS} + b_{ia})(b_{iS} + b_{ib}) &< (b_{iS} + b_{ib} + b_{ia}) \cdot b_{iS} \\
b_{iS}^2 + b_{iS}b_{ib} + b_{ia}b_{iS} + b_{ia}b_{ib} &< b_{iS}^2 + b_{iS}b_{ib} + b_{iS}b_{ia} \\
\cancel{b}_{iS}^2 + \cancel{b}_{iS}b_{ib} + \cancel{b}_{ia}b_{iS} + b_{ia}b_{ib} &< \cancel{b}_{iS}^2 + \cancel{b}_{iS}b_{ib} + \cancel{b}_{iS}b_{ia} \\
b_{ia}b_{ib} &< 0
\end{aligned} \tag{7}$$

This inequality never holds given the constraints on lower bound, $b_{i,j} \geq 1 \quad \forall i, j$, per equation 3. This contradicts our assumption that equation 6 is true, and establishes validity of equation 5. Therefore, given the definition and assumptions of the model, $f^i(S)$ must be submodular. By extension, since $f(S)$ is just a summation of $f^i(S)$ over all i, $f(S)$ is a submodular function [Von10, KG12]. Completing the proof that $f(S)$ is submodular.

□

4 Evaluation of Greedy

We evaluated our submodular model with three TensorFlow job graphs placed on three different physical network with various sizes. All three job graphs are widely used image recognition models with different sizes: (1) VGG19 with 24 nodes [moG17], (2) ResNet50 with 176 nodes [roG17], and (3) Inception-v3 with 311 nodes [toG17a]. Physical network (PN) to place these nodes have CPU and GPU with two to one ratio, i.e., (1) small has three total nodes which consists of two CPU cores and two GPU, (2) medium has 30 total nodes consisting of 20 cores and 10 GPUs, and (3) large has 60 nodes consisting of 40 cores, 20 GPUs. Collected metrics are job completion time (JCT), consumed memory, and CPU time spent to complete the placement. Job graphs and physical network with small, medium, and large sizes were intentionally chosen to compare runtime statistics of the algorithm in addition to JCT. Full source code of our implementation is available at project repository [KCG17].

We compare three algorithms. The first one is our implementation[1] of the TensorFlow greedy heuristic placement algorithm as described in the TensorFlow whitepaper [AAB+15]. The second one is our submodular greedy algorithm described in section 2.2 and the third one is improvements on the submodular greedy described in 2.3. Table 1 shows results for three algorithms run on the small physical topology with three nodes. The first column on each algorithm shows the absolute value of JCT[2] and remaining three values compare performance to the state-of-the-art greedy heuristics (in percentages). Note that table 1 does not capture approximation achieved by the standard greedy algorithm we apply on our submodular model. The table just compares quality of the submodular greedy result, ALG_S, to state-of-the-art greedy heuristic commonly used in practice, ALG_H. In fact, in one of our class assignments we proved standard greedy can not achieve better than 1/2-approximation of the optimal solution achievable with the submodular model: OPT_M. Same note applies to our results of the improved submodular model: ALG_I.

In table 1, the submodular model, *Submodular-G*, JCT varies from extra $34 - 60\%$ without significant memory increase, up to 5% more. Our improved submodular greedy algorithm, *Submodular-GI*, achieves similar JCT as the *Submodular-G* with slightly reduces memory (up to 2.33%). However, the time algorithm spends to complete node placement grows as number of nodes grow. The runtime growth is around 80% for VGG19 with 24 nodes, in order of 10,000% for ResNet50 with 176 nodes, and 38,000% for Inception-v3. This is because *Heuristic-G* considers all devices only once to decide placement of each node and selects the one which completes the operation earliest. However, in order to place each node the submodular models consider every combination of the unassigned nodes and devices, $f^i(S)$, to find the placement with largest marginal value $f(S)$. Such combination makes search space grow exponentially as number of nodes and devices increase.

We observed the same pattern of runtime growth without reducing JCT or memory footprint in our experiments with larger physical topologies with 30 and 60 nodes (table 2 and 3, respectively). On our largest job graph and topology, the submodular algorithms spend around 1,000,000% more time to place 311 nodes on 60 devices. However, submodular model allows us capture additional placement constraints, such as limited device memory and device communication cost when connected nodes are placed across devices. We elaborate on these capabilities in the next section.

	TensorFlow Model											
	VGG19				ResNet50				Inception-v3			
	JCT (μs)	Δ_{JCT}	Δ_{mem}	Δ_{run}	JCT (μs)	Δ_{JCT}	Δ_{mem}	Δ_{run}	JCT (μs)	Δ_{JCT}	Δ_{mem}	Δ_{run}
Heuristic-G	51362816	0%	0%	0%	60986968	0%	0%	0%	49299104	0%	0%	0%
Submodular-G	72795648	+41	+0.99	+84	81836592	+34	+2.73	+10383	78681008	+60	+5	+38698
Submodular-GI	72795648	+41	+0.56	+64	83125360	+36	+2.96	+10408	78802576	+60	+2.77	+38274

Table 1: Comparison of three algorithms on small physical topology with 3 nodes. For each algorithm, collected metrics are job completion time (JCT), Δ_{JCT}: percentage difference of JCT when compared to greedy heuristic (*Heuristic-G*), Δ_{mem}: percentage difference of memory consumption compared to *Heuristic-G*, and Δ_{run}: percentage difference of CPU runtime to complete the placement compared to *Heuristic-G*. Note that *Heuristic-G* has 0% in percentage values as it is being compared to itself (on those cells).

	TensorFlow Model											
	VGG19				ResNet50				Inception-v3			
	JCT (μs)	Δ_{JCT}	Δ_{mem}	Δ_{run}	JCT (μs)	Δ_{JCT}	Δ_{mem}	Δ_{run}	JCT (μs)	Δ_{JCT}	Δ_{mem}	Δ_{run}
Heuristic-G	51362816	0%	0%	0%	60986968	0%	0%	0%	33821568	0%	0%	0%
Submodular-G	54197760	+5.52	-2.40	+1658	77032600	+26	+3.25	+307872	74251264	+119	+5	+976838
Submodular-GI	54197760	+5.52	-2.17	+1643	78322968	+28	+2.24	+301400	73765376	+118	+5	+971220

Table 2: Comparison of three algorithms on medium physical topology with 30 nodes: 20 CPU cores and 10 GPUs.

[1] We had to make our own implementation because the algorithm described in the paper does not exists in the open sourced GitHub repository. Deeper search through TensorFlow mailing list confirmed absence of such implementation [dml17] and states only available scheduling algorithm to be *simple-placer*. *simple-placer* sorts all devices and if no manual placement constraint given by user, *simple-placer* assigns all nodes to the first device on the list [toG17b]. For example, if a physical topology has 20 CPU cores and 10 GPUs, *simple-placer* will assign all nodes to *gpu0* only.

[2] Time cost of operations and its unit (μs) are synthetic. They are chosen based on our estimate, while retaining plausible relative cost of each operation. For algorithm comparison purposes, only relative cost of each operation is important and actual used values do not carry any significance as we use the same cost across all algorithms. See example cost in table 4.

	TensorFlow Model											
	VGG19				ResNet50				Inception-v3			
	JCT (μs)	Δ_{JCT}	Δ_{mem}	Δ_{run}	JCT (μs)	Δ_{JCT}	Δ_{mem}	Δ_{run}	JCT (μs)	Δ_{JCT}	Δ_{mem}	Δ_{run}
Heuristic-G	51362816	0%	0%	0%	60986968	0%	0%	0%	33821568	0%	0%	0%
Submodular-G	51696640	+0.65	+0.87	+2599	79055192	+29	+5.31	+386137	71365584	+111	+1.69	+1088094
Submodular-GI	51696640	+0.65	+0.53	+2578	78237720	+21	+2.34	+393176	71849152	+112	+1.93	+1098507

Table 3: Comparison of three algorithms on large physical topology with 60 nodes: 40 CPU cores and 20 GPUs.

5 Future Work

Greedy heuristic's speed is due to its simplicity. The cost of such simplicity is high when one has to consider device memory capacity and data transmission, important constraints in practice. In fact, a recent work by a team at machine learning company Turi [Inc17] and others reports out-of-memory errors to be common, and developed a technique to trade compute time with device memory to prevent memory usage [CXZG16]. This highlights importance of memory bound optimal placement. Adding inter-device data transmission cost constraint further complicates the problem.

Submodular model can capture memory as well as data transmission constraints. In order to capture different amount of device memory, each device (or group of devices with the same type) can own separate $g(x)$ function, which can be a piece-wise linear function. By carefully choosing this function, we not only can make memory-bound optimal job placement (which prevents out-of-memory error) but also capture memory load-balanced node placement as mentioned as a desirable optimization objective in section 1. For example, we can use a step function, which remains constant until device has available memory and drops significantly down when the device has no more memory available. This forces submodular function $f(S)$ avoid assigning nodes to out-of-memory devices (as marginal benefit is zero).

For memory-balanced placement, the cost of memory can grow as more nodes are assigned to the device (decreasing available memory) forcing function $f(S)$ to choose devices with more memory. A custom $g(x)$ function per device (or devices with the same type) can also model more complex (but practical) behavior of the system. For example, for the servers with different RAM, SSD and HDD capacities, $g(x)$ cost could vary. In practice, these regions can correspond to available memory in RAM, SSD, and HDD, and the variance corresponds to swapping cost to SSD once RAM is saturated, and even more expensive swapping cost to HDD once RAM is also saturated. Such $g(x)$ enables $f(x)$ successfully complete a node assignment (although with slow JCT) instead of failing the assignment when no more RAM is available.

Our current approach to capture data transmission cost is limited to nodes with direct data exchange only. Such transmission cost can be added in the benefit function, such that that cost decreases the benefit of placing a node on a device other than its predecessor, which produces the data to-be-placed node consumes. We would like to further improve our model minimize inter-device data transmission cost, such that if a node can not be co-located with its predecessor, $f(S)$ chooses a device with the smallest transmission cost. We leave implementation and evaluation all these capabilities to future work.

6 Conclusion

In this work, we explored modeling TensorFlow job placement as a submodular function. We formulated placement with shortest job completion time (JCT) as submodular function maximization and implemented the algorithm. Although our evaluation with three different TensorFlow image recognition models and three physical networks does not show improved JCT compared to state-of-the-art greedy heuristic algorithm nor our algorithm achieves better runtime characteristics such as system memory consumption and time spent to place job graph, we see superiority of the submodular model to capture important practical constraints such as memory bound execution, memory-balanced placement, and data transmission cost. We describe these capabilities as an extension to our model and leave their implementation and evaluation to future work.

References

[AAB+15] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp,

Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[CXZG16] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. *CoRR*, abs/1604.06174, 2016.

[dml17] Tensorflow developer mailing list. Greedy heuristic algorithm implementation does not exist, 2017. [Available at https://groups.google.com/a/tensorflow.org/forum/#!msg/discuss/TxRQom4fq9o/nj_tkWKmFwAJ, Accessed 30-April-2017].

[Inc17] Turi Inc. Fast, scalable machine learning modeling in python, 2017. [Available at https://turi.com/, Accessed 30-April-2017].

[KCG17] Nodir Kodirov, Amanda Carbonari, and Maximilian Golub. 536n project github repository, frozen under release name 'project-report', 2017. [Available at https://github.com/knodir/tensorflow-sm/releases/tag/project-report, Accessed 30-April-2017].

[KG12] Andreas Krause and Daniel Golovin. Submodular function maximization, 2012.

[moG17] @machrisaa on GitHub. Vgg19 and vgg16 on tensorflow, 2017. Available at https://github.com/machrisaa/tensorflow-vgg, Accessed 30-April-2017.

[roG17] @ry on GitHub. Resnet in tensorflow, 2017. Available at https://github.com/ry/tensorflow-resnet, Accessed 30-April-2017.

[toG17a] @tensorflow on GitHub. Inception in tensorflow, 2017. Available at https://github.com/tensorflow/models/tree/master/inception, Accessed 30-April-2017.

[toG17b] @tensorflow on GitHub. simple-placer algorithm, 2017. [Available at https://github.com/tensorflow/tensorflow/blob/1a1527ab1fcffd94f692b1301fdbe87903ce7bdb/tensorflow/core/common_runtime/simple_placer.cc#L749, Accessed 30-April-2017].

[Von10] Jan Vondrak. Submodular functions, 2010.

[Wik17] Wikipedia. Job shop scheduling, 2017. Available at https://en.wikipedia.org/w/index.php?title=Job_shop_scheduling&oldid=777696589, Accessed 30-April-2017.

6.1 Appendix

This walk-though is intended to provide intuition behind submodularity functions. Figure 1a shows the node graph with 5 nodes. We place this graph on 3 devices shown in Figure 4. We refer to core1 as device 1, core2 as device 2, and gpu1 as device 3. Table 4 shows cost of running operations on these devices.

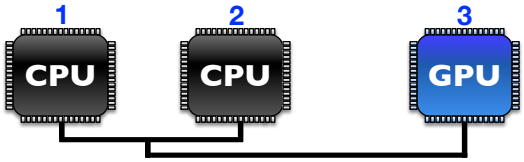

Figure 3: Two CPU cores and a GPU core available for node placement.

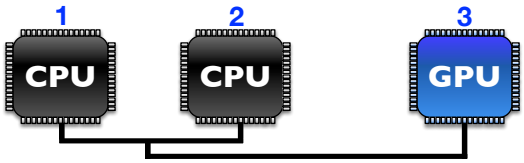

Figure 4: Two CPU cores and a GPU core available for node placement.

Operation	Cost on CPU	Cost on GPU
Convolution (CV)	4	2
MaxPool (MP)	6	5
AvgPool (AP)	1	3
Concat (CC)	5	7

Table 4: Time cost (s) of executing an operation on CPU and GPU.

	device_1 (CPU1)	device_2 (CPU2)	device_3 (GPU)
node_1	4	4	2
node_2	6	6	5
node_3	1	1	3
node_4	4	4	2
node_5	5	5	7

Table 5: Time cost (s) of executing a node j on a device i.

Based on the costs outlined in Table 4, we construct the $c_{j,i}$ values in Table 5. The table is indexed starting at 1, so node_1 refers to row 1 and device_1 refers to column 1. We use all functions defined in Section 2.2. Submodular maximization will choose an S to evaluate with f, we will walk through our example using the greedy algorithm (i.e., start with $\emptyset$ and greedily add elements with the highest $f(S)$). We will describe each iteration of the greedy algorithm as "rounds". In each round greedy will choose the node and device placement which produces the highest $f(S)$ value.

Round 1:

In this first round, greedy is choosing to place the first node. Typically this will cause greedy to place the lowest cost node, as it provides the best benefit. It does not and cannot consider parallelism, because no other nodes have been placed. Therefore, any node it is looking at adding, cannot be parallel with the already placed nodes. Therefore the dependency list for every set in this round is [{}].The chosen S is E_{13} because it has the maximum $f(S)$ value of 1.41.

$S = \{E_{11}\}$	$S = \{E_{12}\}$	$S = \{E_{13}\}$	$S = \{E_{14}\}$	$S = \{E_{15}\}$
$c_{1,1} = 4$	$c_{1,2} = 6$	$c_{1,3} = 1$	$c_{1,4} = 4$	$c_{1,5} = 5$
$b_{1,1} = (2 - \left(\frac{4+0}{100}\right)) \cdot 1$ $= 2 - 0.04$ $= 1.96$	$b_{1,2} = 1.94$	$b_{1,3} = 1.99$	$b_{1,4} = 1.96$	$b_{1,5} = 1.95$
$f^1(S) = \sqrt{1.96}$	$f^1(S) = \sqrt{1.94}$	$f^1(S) = \sqrt{1.99}$	$f^1(S) = \sqrt{1.96}$	$f_1(S) = \sqrt{1.95}$
$f(S) = 1.400$	$f(S) = 1.392$	$\boldsymbol{f(S) = 1.410}$	$f(S) = 1.400$	$f(S) = 1.396$

$S = \{E_{21}\}$	$S = \{E_{22}\}$	$S = \{E_{23}\}$	$S = \{E_{24}\}$	$S = \{E_{25}\}$
$c_{2,1} = 4$	$c_{2,2} = 6$	$c_{2,3} = 1$	$c_{2,4} = 4$	$c_{2,5} = 5$
$b_{2,1} = 1.96$	$b_{2,2} = 1.94$	$b_{2,3} = 1.99$	$b_{2,4} = 1.96$	$b_{2,5} = 1.95$
$f^2(S) = \sqrt{1.96}$	$f^2(S) = \sqrt{1.94}$	$f^2(S) = \sqrt{1.99}$	$f^2(S) = \sqrt{1.96}$	$f^2(S) = \sqrt{1.95}$
$f(S) = 1.400$	$f(S) = 1.392$	$f(S) = 1.410$	$f(S) = 1.400$	$f(S) = 1.396$

$S = \{E_{31}\}$	$S = \{E_{32}\}$	$S = \{E_{33}\}$	$S = \{E_{34}\}$	$S = \{E_{35}\}$
$c_{3,1} = 2$	$c_{3,2} = 5$	$c_{3,3} = 3$	$c_{3,4} = 2$	$c_{3,5} = 7$
$b_{3,1} = 1.98$	$b_{3,2} = 1.95$	$b_{3,3} = 1.97$	$b_{3,4} = 1.98$	$b_{3,5} = 1.93$
$f^3(S) = \sqrt{1.98}$	$f^3(S) = \sqrt{1.95}$	$f^3(S) = \sqrt{1.97}$	$f^3(S) = \sqrt{1.98}$	$f^3(S) = \sqrt{1.93}$
$f(S) = 1.407$	$f(S) = 1.396$	$f(S) = 1.403$	$f(S) = 1.407$	$f(S) = 1.389$

Round 2:

In this round, parallelism comes into play. As you can see in the graph topology, Figure 1, node 3 and 4 can be parallel. It is important to note that the set $\{E_{13}, E_{14}\}$ has a parallelism factor of 1. This is due to how we defined parallelism in 2.2. We define it as all the nodes j is parallel to and are NOT placed on device i. Therefore, the parallelism factor of $\{E_{13}, E_{14}\}$ is 1. Here the greedy algorithm chooses the set $\{E_{13}, E_{34}\}$ because it has the value 3.399.

$S = \{E_{13}, E_{11}\}$
$c_{1,1} = 4$
$D_S = [\{\}, \{1\}]$
$b_{1,1} = (2 - \left(\frac{4+1}{100}\right)) \cdot 1$
$= 2 - 0.05 = 1.95$
$f^1(S) = \sqrt{1.99 + 1.95} = 1.984$
$f(S) = 1.984$

$S = \{E_{13}, E_{12}\}$
$c_{1,2} = 6$
$D_S = [\{\}, \{1\}]$
$b_{1,2} = (2 - \left(\frac{6+1}{100}\right)) \cdot 1$
$= 2 - 0.07 = 1.93$
$f^1(S) = \sqrt{1.99 + 1.93} = 1.979$
$f(S) = 1.979$

$S = \{E_{13}, E_{14}\}$
$c_{1,4} = 4$
$D_S = [\{\}, \{\}]$
$b_{1,4} = (2 - \left(\frac{4+1}{100}\right)) \cdot 1$
$= 2 - 0.05 = 1.95$
$f^1(S) = \sqrt{1.99 + 1.95} = 1.984$
$f(S) = 1.984$

$S = \{E_{13}, E_{15}\}$
$c_{1,5} = 5$
$D_S = [\{\}, \{1\}]$
$b_{1,5} = (2 - \left(\frac{5+1}{100}\right)) \cdot 1$
$= 2 - 0.06 = 1.94$
$f^1(S) = \sqrt{1.99 + 1.94} = 1.982$
$f(S) = 1.982$

$S = \{E_{13}, E_{21}\}$
$c_{2,1} = 4$
$D_S = [\{\}, \{1\}]$
$b_{2,1} = (2 - \left(\frac{4+0}{100}\right)) \cdot 1$
$= 2 - 0.04 = 1.96$
$f^1(S) = \sqrt{1.99}$
$f^2(S) = \sqrt{1.96}$
$f(S) = f^1(S) + f^2(S) = 2.810$

$S = \{E_{13}, E_{22}\}$
$c_{2,2} = 6$
$D_S = [\{\}, \{1\}]$
$b_{2,2} = (2 - \left(\frac{6+0}{100}\right)) \cdot 1$
$= 2 - 0.06 = 1.94$
$f^1(S) = \sqrt{1.99}$
$f^2(S) = \sqrt{1.94}$
$f(S) = f^1(S) + f^2(S) = 2.806$

$S = \{E_{13}, E_{24}\}$
$c_{2,4} = 4$
$D_S = [\{\}, \{\}]$
$b_{2,4} = (2 - \left(\frac{4+0}{100}\right)) \cdot 2$
$= (2 - 0.04) \cdot 2 = 3.92$
$f^1(S) = \sqrt{1.99}$
$f^2(S) = \sqrt{3.92}$
$f(S) = f^1(S) + f^2(S) = 3.389$

$S = \{E_{13}, E_{25}\}$
$c_{2,5} = 5$
$D_S = [\{\}, \{1\}]$
$b_{2,5} = (2 - \left(\frac{5+0}{100}\right)) \cdot 1$
$= 2 - 0.05 = 1.95$
$f^1(S) = \sqrt{1.99}$
$f^2(S) = \sqrt{1.95}$
$f(S) = f^1(S) + f^2(S) = 2.806$

$S = \{E_{13}, E_{31}\}$
$c_{3,1} = 2$
$D_S = [\{\}, \{1\}]$
$b_{3,1} = (2 - \left(\frac{2+0}{100}\right)) \cdot 1$
$= 2 - 0.02 = 1.98$
$f^1(S) = \sqrt{1.99}$
$f^3(S) = \sqrt{1.98}$
$f(S) = f^1(S) + f^3(S) = 2.817$

$S = \{E_{13}, E_{32}\}$
$c_{3,2} = 5$
$D_S = [\{\}, \{1\}]$
$b_{3,2} = (2 - \left(\frac{5+0}{100}\right)) \cdot 1$
$= 2 - 0.05 = 1.95$
$f^1(S) = \sqrt{1.99}$
$f^3(S) = \sqrt{1.95}$
$f(S) = f^1(S) + f^3(S) = 2.806$

$S = \{E_{13}, E_{34}\}$
$c_{3,4} = 2$
$D_S = [\{\}, \{\}]$
$b_{3,4} = (2 - \left(\frac{2+0}{100}\right)) \cdot 2$
$= (2 - 0.02) \cdot 2 = 3.96$
$f^1(S) = \sqrt{1.99}$
$f^3(S) = \sqrt{3.96}$
$\boldsymbol{f(S) = f^1(S) + f^3(S) = 3.399}$

$S = \{E_{13}, E_{35}\}$
$c_{3,5} = 7$
$D_S = [\{\}, \{1\}]$
$b_{3,5} = (2 - \left(\frac{7+0}{100}\right)) \cdot 1$
$= 2 - 0.07 = 1.93$
$f^1(S) = \sqrt{1.99}$
$f^3(S) = \sqrt{1.93}$
$f(S) = f^1(S) + f^3(S) = 2.799$

Round 3:

Now that the two parallel nodes have been placed, we do not display the dependency lists as they are not needed, all other placements will have a parallelism factor of 1. In this round, you can see how the algorithm favors placing nodes on an empty device, all nodes placed on the empty device 2, have a much higher value than the rest of the nodes on other devices. This causes the greedy algorithm to choose E_{21} as the next placement.

$S = \{E_{13}, E_{34}, E_{11}\}$
$c_{1,1} = 4$
$b_{1,1} = (2 - \left(\frac{4+1}{100}\right)) \cdot 1$
$= 2 - 0.05 = 1.95$
$f^1(S) = \sqrt{1.99 + 1.95} = 1.984$
$f^3(S) = \sqrt{3.96} = 1.989$
$f(S) = f^1(S) + f^3(S) = 3.973$

$S = \{E_{13}, E_{34}, E_{12}\}$
$c_{1,2} = 6$
$b_{1,2} = (2 - \left(\frac{6+1}{100}\right)) \cdot 1$
$= 2 - 0.07 = 1.93$
$f^1(S) = \sqrt{1.99 + 1.93} = 1.979$
$f^3(S) = \sqrt{3.96} = 1.989$
$f(S) = f^1(S) + f^3(S) = 3.968$

$S = \{E_{13}, E_{34}, E_{15}\}$
$c_{1,5} = 5$
$b_{1,5} = (2 - \left(\frac{2+1}{100}\right)) \cdot 1$
$= 2 - 0.03 = 1.97$
$f^1(S) = \sqrt{1.99 + 1.97} = 1.989$
$f^3(S) = \sqrt{3.96} = 1.989$
$f(S) = f^1(S) + f^3(S) = 3.978$

$S = \{E_{13}, E_{34}, E_{21}\}$
$c_{2,1} = 4$
$b_{2,1} = 2 - \left(\frac{4+0}{100}\right) \cdot 1$
$= 2 - 0.04 = 1.96$
$f^1(S) = \sqrt{1.99} = 1.410$
$f^2(S) = \sqrt{1.96} = 1.400$
$f^3(S) = \sqrt{3.96} = 1.989$
$\boldsymbol{f(S) = \sum_{i=1}^{3} f^i(S) = 4.799}$

$S = \{E_{13}, E_{34}, E_{22}\}$
$c_{2,2} = 6$
$b_{2,2} = 2 - \left(\frac{6+0}{100}\right) \cdot 1$
$= 2 - 0.06 = 1.94$
$f^1(S) = \sqrt{1.99} = 1.410$
$f^2(S) = \sqrt{1.94} = 1.392$
$f^3(S) = \sqrt{3.96} = 1.989$
$f(S) = \sum_{i=1}^{3} f^i(S) = 4.791$

$S = \{E_{13}, E_{34}, E_{25}\}$
$c_{2,5} = 5$
$b_{2,5} = (2 - \left(\frac{5+0}{100}\right)) \cdot 1$
$= 2 - 0.05 = 1.95$
$f^1(S) = \sqrt{1.99} = 1.410$
$f^2(S) = \sqrt{1.95} = 1.396$
$f^3(S) = \sqrt{3.96} = 1.989$
$f(S) = \sum_{i=1}^{3} f^i(S) = 4.795$

$S = \{E_{13}, E_{34}, E_{31}\}$
$c_{3,1} = 2$
$b_{3,1} = (2 - \left(\frac{2+2}{100}\right)) \cdot 1$
$= 2 - 0.04 = 1.96$
$f^1(S) = \sqrt{1.99} = 1.410$
$f^3(S) = \sqrt{3.96 + 1.96} = 2.433$
$f(S) = f^1(S) + f^3(S) = 3.843$

$S = \{E_{13}, E_{34}, E_{32}\}$
$c_{3,2} = 5$
$b_{3,2} = (2 - \left(\frac{5+2}{100}\right)) \cdot 1$
$= 2 - 0.07 = 1.93$
$f^1(S) = \sqrt{1.99} = 1.410$
$f^3(S) = \sqrt{3.96 + 1.93} = 2.426$
$f(S) = f^1(S) + f^3(S) = 3.836$

$S = \{E_{13}, E_{34}, E_{35}\}$
$c_{3,5} = 7$
$b_{3,5} = (2 - \left(\frac{7+2}{100}\right)) \cdot 1$
$= 2 - 0.09 = 1.91$
$f^1(S) = \sqrt{1.99} = 1.410$
$f^3(S) = \sqrt{3.96 + 1.91} = 2.422$
$f(S) = f^1(S) + f^3(S) = 3.832$

Round 4:

Now that every device has a node placed on it, the algorithm prioritizes placement on devices with the lowest total node cost. This is captured in the $placed\_nodes_i$ value, which is just the summation of all $c_{i,j'}$ where i is the device that this node is currently being placed on. Since E_{13} has the lowest cost of the placed nodes and node 5 has the lowest cost of the non-placed nodes, greedy chooses E_{15} as the next placement.

$S = \{E_{13}, E_{34}, E_{21}, E_{12}\}$
$c_{1,2} = 6$
$b_{1,2} = (2 - \left(\frac{6+1}{100}\right)) \cdot 1$
$= 2 - 0.07$
$= 1.93$
$f^1(S) = \sqrt{1.99 + 1.93} = 1.979$
$f^2(S) = \sqrt{1.96} = 1.400$
$f^3(S) = \sqrt{3.96} = 1.989$
$f(S) = 1.979 + 1.400 + 1.989 = 5.368$

$S = \{E_{13}, E_{34}, E_{21}, E_{15}\}$
$c_{1,5} = 5$
$b_{1,5} = (2 - \left(\frac{5+1}{100}\right)) \cdot 1$
$= 2 - 0.06$
$= 1.94$
$f^1(S) = \sqrt{1.99 + 1.94} = 1.982$
$f^2(S) = \sqrt{1.96} = 1.400$
$f^3(S) = \sqrt{3.96} = 1.989$
$\boldsymbol{f(S) = 1.982 + 1.400 + 1.989 = 5.371}$

$S = \{E_{13}, E_{34}, E_{21}, E_{22}\}$
$c_{2,2} = 6$
$b_{1,2} = (2 - \left(\frac{6+4}{100}\right)) \cdot 1$
$= 2 - 0.1$
$= 1.9$
$f^1(S) = \sqrt{1.99} = 1.410$
$f^2(S) = \sqrt{1.96 + 1.9} = 1.964$
$f^3(S) = \sqrt{3.96} = 1.989$
$f(S) = 1.410 + 1.964 + 1.989 = 5.363$

$S = \{E_{13}, E_{34}, E_{21}, E_{25}\}$
$c_{2,5} = 5$
$b_{1,5} = (2 - \left(\frac{5+4}{100}\right)) \cdot 1$
$= 2 - 0.11$
$= 1.89$
$f^1(S) = \sqrt{1.99} = 1.410$
$f^2(S) = \sqrt{1.96 + 1.89} = 1.962$
$f^3(S) = \sqrt{3.96} = 1.989$
$f(S) = 1.410 + 1.962 + 1.989 = 5.361$

$S = \{E_{13}, E_{34}, E_{21}, E_{32}\}$
$c_{3,2} = 5$
$b_{3,2} = (2 - \left(\frac{5+2}{100}\right)) \cdot 1$
$= 2 - 0.07$
$= 1.93$
$f^1(S) = \sqrt{1.99} = 1.410$
$f^2(S) = \sqrt{1.96} = 1.4$
$f^3(S) = \sqrt{1.93 + 3.96} = 2.426$
$f(S) = 1.41 + 1.4 + 2.426 = 5.236$

$S = \{E_{13}, E_{34}, E_{21}, E_{35}\}$
$c_{3,5} = 7$
$b_{3,5} = (2 - \left(\frac{7+2}{100}\right)) \cdot 1$
$= 2 - 0.09$
$= 1.91$
$f^1(S) = \sqrt{1.99} = 1.410$
$f^2(S) = \sqrt{1.96} = 1.4$
$f^3(S) = \sqrt{1.91 + 3.96} = 2.422$
$f(S) = 1.41 + 1.4 + 2.422 = 5.232$

Round 5:

Finally, greedy must place the last node. It follows the same methodology as explained before and attempts to place it on the device with the least total cost. Here is where the parallelism factor fails. Because it over inflates the f_i value, it ends up forcing greedy to make a worse placement. We detail a solution to this problem in the next section (Section 5. The final placement is $E = \begin{Bmatrix} 0 & 4 & 0 \\ 0 & 6 & 0 \\ 1 & 0 & 0 \\ 0 & 0 & 2 \\ 5 & 0 & 0 \end{Bmatrix}$. Which gives a total node Completion Time (JCT) of $4 + 6 + 2 + 5 = 17$, based on the calculation outlined in Section 2.2

$S = \{E_{13}, E_{34}, E_{21}, E_{15}, E_{12}\}$
$c_{1,2} = 6$
$b_{1,2} = (2 - \left(\frac{6+1+5}{100}\right)) \cdot 1$
$= 2 - 0.12$
$= 1.88$
$f^1(S) = \sqrt{1.99 + 1.94 + 1.88} = 2.410$
$f^2(S) = \sqrt{1.96} = 1.4$
$f^3(S) = \sqrt{3.96} = 1.989$
$f(S) = 2.410 + 1.400 + 1.989 = 5.799$

$S = \{E_{13}, E_{34}, E_{21}, E_{15}, E_{22}\}$
$c_{2,2} = 6$
$b_{2,2} = (2 - \left(\frac{6+4}{100}\right)) \cdot 1$
$= 2 - 0.1$
$= 1.9$
$f^1(S) = \sqrt{1.99 + 1.94} = 1.982$
$f^2(S) = \sqrt{1.96 + 1.9} = 1.964$
$f^3(S) = \sqrt{3.96} = 1.989$
$\boldsymbol{f(S) = 1.982 + 1.964 + 1.989 = 5.935}$

$S = \{E_{13}, E_{34}, E_{21}, E_{15}, E_{32}\}$
$c_{3,2} = 5$
$b_{3,2} = (2 - \left(\frac{5+2}{100}\right)) \cdot 1$
$= 2 - 0.07$
$= 1.93$
$f^1(S) = \sqrt{1.99 + 1.94} = 1.982$
$f^2(S) = \sqrt{1.96} = 1.4$
$f^3(S) = \sqrt{3.96 + 1.93} = 2.426$
$f(S) = 1.982 + 1.400 + 2.426 = 5.808$