Critical path estimation in heterogeneous scheduling heuristics

Thomas McSweeney*

7th August 2020

1 Introduction

Recall from Chapter X that the *Heterogeneous Earliest Finish Time* (HEFT) heuristic prioritizes all tasks t_i, $i = 1, \ldots, n$, by recursively computing a corresponding sequence of numbers u_i which are intended to represent *critical path* lengths from each task to the end. As noted previously, however, the concept of the *critical path* is not clearly defined for heterogeneous target platforms: DAG weights are not fixed at this stage so there are multiple ways we could define a *longest* (i.e., *costliest*) path. Consider for example the simple DAG shown in Figure 1, where the labels represent all the possible weights each task/edge may take on a two-processor target platform; specifically, the labels (W_i^1, W_i^2) near the nodes represent the computation costs on processors $P1$ and $P2$, respectively, while the edge labels $(0 = W_{ik}^{11} = W_{ik}^{22}, W_{ik}^{12}, W_{ik}^{21})$ represent the possible communication costs. What is the longest path through this graph?

It is not obvious how this should be defined. The HEFT approach, as described in previous chapters, is to use mean values over all sets of possible costs to fix the DAG weights and then compute the critical path lengths in a standard dynamic programming manner. For the example we find that

$$u_6 = 2.5, u_5 = 7, u_4 = 7.75$$
$$u_3 = 16, u_2 = 5.75, u_1 = 16.25, u_0 = 24.75,$$

giving a scheduling priority list of $\{0, 1, 3, 4, 5, 2, 6\}$. The schedule length obtained by HEFT with these priorities is 22, so note in particular that u_0, the rank of the single entry task, is not a lower bound on this value, unlike in the homogeneous case. The question then is, what quantity do the u_i values actually represent? The most intuitive interpretation is perhaps that the HEFT ranks are estimates

*School of Mathematics, University of Manchester, Manchester, M13 9PL, England (thomas.mcsweeney@postgrad.manchester.ac.uk).

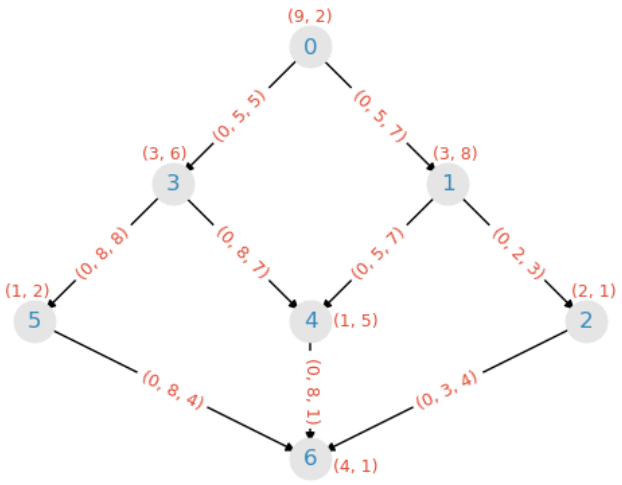

Figure 1: Simple task graph with costs on a two-processor target platform.

of what the critical paths are *likely* to be but there is no robust mathematical justification for believing that such a definition is truly the most useful.

Given this ambiguity, alternative ways to define the critical path in HEFT have been considered before, most notably by Zhao and Sakellariou [8], who empirically compared the performance of HEFT when averages other than the mean (e.g., median, maximum, minimum) are used to compute upward (or downward) ranks. Their conclusions were that using the mean is not clearly superior to other averages, although none of the other options were consistently better. Indeed, perhaps the biggest takeaway from their investigation was that HEFT is very sensitive to how priorities are computed, with significant variation being seen for different graphs and target platforms. In this chapter we undertake a similar investigation with the aim of establishing if there are choices which do consistently outperform the standard HEFT task ranking phase.

This will be an empirically-driven study, as is common in this area. To facilitate this investigation we created a software package that simulates heterogeneous scheduling problems, much like that described in the previous chapter, although not restricted to accelerated target platforms. As before, the (`Python`) source code for this simulator can be found on Github[1] and all of the results presented here can be re-run from scripts contained therein.

2 Optimistic bounds

Functionally, the critical path is used in HEFT as a lower bound on the makespan, so that minimizing the critical path gives us the most scope to minimize the

[1] https://github.com/mcsweeney90/critical-path-estimation

Table 1: Upward and optimistic ranks.

Task	Upward rank	Optimistic rank
0	24.75	16
1	16.25	8
2	5.75	2
3	16	8
4	7.75	5
5	7	3
6	2.5	1

makespan (assuming we make good use of our parallel resources). With this in mind, there are many different ways we can define the critical path so that it gives a lower bound on the makespan of any possible schedule. The most straightforward approach would be to just set all weights to their minimal values but a tighter bound can be computed in the following manner. First, define O_i^a for all tasks t_i and processors p_a to be the critical path length from t_i to the end (inclusive), assuming that it is scheduled on processor p_a. These values can easily be computed recursively by setting $O_i^a = W_i^a \ \forall a$ for all exit tasks then moving up the DAG and setting

$$O_i^a = W_i^a + \max_{k \in S_i} \left(\min_{b=1,\ldots,q} \left(O_k^b + W_{ik}^{ab} \right) \right) \quad \forall a \tag{1}$$

for all other tasks. Then for each $i = 1, \ldots, n$,

$$O_i = \min_{a=1,\ldots,q} O_i^a \tag{2}$$

gives a true lower bound on the remaining cost of any schedule once the execution of task t_i begins. These O_i values could be useful as alternative task priorities in HEFT, especially since the cost of computing all of the O_i^a in this manner is only $O(m+n) \approx O(n^2)$ so in particular is no more expensive than the usual HEFT prioritization phase. For example, for the simple graph shown in Figure 1, we find that the O_i values are as given in Table 1 (with the u_i included for comparison). Interestingly, we see that tasks 1 and 3 have the same optimistic rank (8) and the performance of the alternative ranks relative to the standard u_i sequence in HEFT depends on which is chosen to be scheduled first; if Task 1, the priority list does not change so the schedule makespan is 22, but if Task 3 is selected instead, the final schedule makespan is smaller than the original at 20.

Of course, this is only one example: it should be emphasized here that there is absolutely no mathematically valid reason to suppose that using the O_i sequence

instead of u_i as the task ranks in HEFT will actually lead to superior performance in general. Still, it seems worthwhile to investigate this empirically using our simulator, which we do in Section 4.

(Note that the optimistic critical path defined here is extremely similar to the optimistic cost used in the PEFT heuristic; this will be discussed further in Section 5.)

3 Stochastic interpretation

In this section we propose a family of alternative task ranking phases in HEFT based on the following interpretation of the standard ranking phase. First, note that by using average values over all sets of possible task and edge costs, HEFT is implicitly assuming that any member of any set is just as likely to be incurred as any other; conceptually, HEFT is attempting to account for the uncertainty of the processor selection phase by assuming that for any given task all processors are equally likely to ultimately be chosen. So, effectively, at the prioritization phase HEFT views the node and edge weights as independent discrete random variables (RVs) with associated probability mass functions (pmfs) given by the aforementioned assumption. More precisely, let m_i be the pmf corresponding to the task weight variable w_i and m_{ik} that for the edge weight w_{ik}, then

$$m_i(W_i^a) \coloneqq \mathbb{P}[w_i = W_i^a] = \frac{1}{n_p} \quad \forall a$$

and

$$\begin{aligned} m_{ik}(W_{ik}^{ab}) &= m_i(W_i^a) \cdot m_k(W_k^b) \\ &= \frac{1}{n_p^2} \quad \forall a, b. \end{aligned}$$

Note that the expected values of the node and edge weights are therefore given by

$$\mathbb{E}[w_i] = \sum_{\ell \in L_i} \ell m_i(\ell) = \frac{1}{n_p} \sum_a W_i^a \tag{3}$$

$$\mathbb{E}[w_{ik}] = \sum_{\ell \in L_{ik}} \ell m_{ik}(\ell) = \frac{1}{n_p^2} \sum_{a,b} W_{ik}^{ab}. \tag{4}$$

In particular, this means that $\mathbb{E}[w_i] = \overline{w_i}$ and $\mathbb{E}[w_{ik}] = \overline{w_{ik}}$ so that the computation of the upward ranks u_i can instead be done by setting $u_i = \mathbb{E}[w_i]$ for all exit tasks, then moving up the DAG and recursively computing

$$u_i = \mathbb{E}[w_i] + \max_{k \in S_i} \big(u_k + \mathbb{E}[w_{ik}] \big) \tag{5}$$

for all other tasks.

In summary, since all possible node and edge weights are known but their actual values at runtime aren't (at least without restricting the processor selection phase), HEFT estimates critical path lengths from all tasks in a task graph G through a two-step process:

1. An associated graph G_s—referred to as *stochastic* because all of its weights are RVs—is implicitly constructed with node and edge pmfs m_i and m_{ik} as defined above.

2. The numbers u_i are recursively computed for all tasks in G_s using (5), and taken as the critical path lengths from the corresponding tasks in G.

In the following two sections, we propose modifications of both steps so as to obtain different critical path estimates that may be used as task ranks in HEFT. The performance of these will then be evaluated through extensive numerical simulations in Section 4.

3.1 The critical path of G_s

A natural question arises from the interpretation outlined in the previous section: what is the relationship between the sequence of numbers u_i as defined by (5) and the critical path of the stochastic graph G_s? (Of course, since all of the weights are RVs, the critical path of G_s is itself stochastic.) In fact, it has long been known in the context of *Program Evaluation and Review Technique* (PERT) network analysis that the numbers u_i are *lower bounds* on the expected value of the critical path lengths of the stochastic DAG. This result dates back at least as far as Fulkerson [4], who referred to it as already being widely-known and gave a simple proof. This prompts another question: does using the actual expected values of the critical path lengths as the task priories in HEFT lead to superior performance?

Unfortunately, computing the moments of the critical path length of a graph whose weights are discrete RVs was shown to be a $\#P$-complete problem by Hagstrom [5]. This means that it is generally impractical to compute the true expected values. However, efficient methods which yield better approximations than the u_i numbers are known; we discuss examples in the following two sections.

3.1.1 Monte Carlo sampling

Monte Carlo (MC) methods have a long history as a means of approximating the longest path distribution for PERT networks, dating back to at least the early 1960s [7]. The idea is to simulate the realization of all RVs (according to their pmfs) and then evaluate the critical path of the resulting deterministic graph. This is done repeatedly, giving a set of critical path instances whose

empirical distribution function is guaranteed to converge to the true distribution by the Glivenko-Cantelli theorem [1]. Furthermore, analytical results allow us to quantify the approximation error for any given the number of realizations—and therefore the number of realizations needed to reach a desired accuracy—although we do not make use of such results here.

Table X illustrates how our estimate of the expected critical path of the stochastic graph in Figure X evolves as the number of samples increases...

The downside of Monte Carlo sampling is its cost, particularly when the number of realizations required is large, although modern architectures are well-suited to this approach because of their parallelism so this problem may no longer be as acute as it once was.

3.1.2 Fulkerson's bound

For all $i = 1, \ldots, n$, let c_i be the critical path length from task t_i to the end and let $e_i = \mathbb{E}[c_i]$ be its expected value. In addition to proving that the u_i sequence defines lower bounds on the critical path lengths, i.e., $u_i \leq e_i \ \forall i$, Fulkerson also showed how an alternative sequence of numbers which give tighter bounds can easily be constructed.

First we assume that G_s is expressed in an equivalent formulation without node weights. The most straightforward way to do this is to simply redefine the edge weights so that they also include the computation cost of the parent task and, if the child task is an exit, the computation cost of the child as well. More precisely, we define a new set of possible edge weights $\tilde{L}_{ik}$ by

$$\tilde{L}_{ik} = \{\tilde{W}_{ik}^{ab} \coloneqq W_i^a + \delta_k W_k^b + W_{ik}^{ab}\}_{a,b=1,\ldots,q},$$

where $\delta_k = 1$ if t_k is an exit task and zero otherwise. We also define a new edge weight variable $\tilde{w}_{ik} \in \tilde{L}_{ik}$ and new edge pmfs $\tilde{m}_{ik}$ for which $\tilde{m}(\tilde{W}_{ik}^{ab}) \equiv m(W_{ik}^{ab})$. Note that removing the node weights is not strictly necessary but simply makes the elucidation cleaner so we should emphasize that all of the following still holds, with only minor adjustments, if this is not done.

Now, for $i = 1, \ldots, n$, define Z_i to be the set of all weight RVs corresponding to edges downward of t_i (i.e., the remainder of the graph). Let R_i be the set of all possible *realizations* of the RVs in Z_i. Given a realization $z_i \in R_i$, let $\ell(z_i)$ be the critical path length from task t_i to the end. Then by the definition of the expected value we have

$$e_i = \sum_{z_i \in R_i} \mathbb{P}[Z_i = z_i]\ell(z_i). \tag{6}$$

Let $C_i \coloneqq \{w_{ik}\}_{k \in S_i}$ be the set of all the weight RVs corresponding to edges which connect task t_i to its children. Note that

$$Z_i = C_i \cup_{k \in S_i} Z_k$$

and any realization $z_i \in R_i$ can therefore be expressed as $z_i = c_i \cup_{k \in S_i} z_k^i$, where $z_k^i \in R_k$ and $c_i = \{z_{ik}^i\}_{k \in S_i}$ is the set of realizations of the edge weight RVs in C_i. In particular, this means that we can write

$$\ell(z_i) = \max_{k \in S_i} \{\ell(z_k^i) + z_{ik}^i\}.$$

Furthermore, by the independence assumptions made, we have that

$$\mathbb{P}[Z_i = z_i] = \mathbb{P}[C_i = c_i] \prod_{k \in S_i} \mathbb{P}[Z_k = z_i^k].$$

This means that we can rewrite equation (6) as

$$\begin{aligned} e_i &= \sum_{z_i \in R_i} \mathbb{P}[Z_i = z_i] \ell(z_i) \\ &= \sum_{c_i} \sum_{\substack{z_k \in R_k, \\ k \in S_i}} \mathbb{P}[C_i = c_i] \mathbb{P}[Z_k = z_k] \max_{k \in S_i} \{\ell(z_k) + z_{ik}\}, \end{aligned} \tag{7}$$

where z_{ik} is the realization of the edge weight RV w_{ik} defined by the set of realizations z_k.

It is relatively straightforward to show that the identity $u_i \leq e_i$ holds by manipulating equation (7); the reader is directed to Fulkerson's original paper for details [4]. Moreover, suppose we define a sequence of numbers by $f_i = 0$, if t_i is an exit task, and

$$\begin{aligned} f_i &= \sum_{z_i \in R_i} \mathbb{P}[Z_i = z_i] \max_{k \in S_i} \{f_k + z_{ik}\} \\ &= \sum_{c_i} \sum_{\substack{z_k \in R_k, \\ k \in S_i}} \mathbb{P}[C_i = c_i] \mathbb{P}[Z_k = z_k] \max_{k \in S_i} \{f_k + z_{ik}\}, \end{aligned} \tag{8}$$

for all other tasks, then Fulkerson showed that $u_i \leq f_i \leq e_i$ also holds—i.e, the f_i give a tighter bound on the expected values of the critical path lengths.

To compute each of the f_i using (8) we need to do an awful lot of work: suppose t_i has K children, then for any one of them t_k we need to do $O(|\tilde{L}_{ik}|)$ operations, i.e., $O(|\tilde{L}_{ik}|^K)$ in total. In general, $|\tilde{L}_{ik}|$ can be $O(p^2)$ and K can be $O(n^2)$ so computing the f_i in this manner is not always practical. Fortunately, a more efficient method was given by Clingen [2] in the context of extending Fulkerson's method to the case where edge weights are modeled as continuous random variables, although here we follow the slightly more compact notation of Elmaghraby [3].

It is well-known that the cumulative probability mass function of the maximum of a finite set of discrete RVs is equal to the product of the individual cumulative pmfs of the RVs. Let M_{ik} be the cumulative pmf along edge (t_i, t_k),

so that $M_{ik}(x) = \mathbb{P}[\tilde{w}_{ik} \leq x]$. Define the related function $M^*_{ik}(x) = \mathbb{P}[\tilde{w}_{ik} < x]$. Let Z_i be the set of all possible values of $f_k + \tilde{w}_{ik}$, for $k \in S_i$, and let z run over all elements of Z_i. For $i = 1, \ldots, n$, define

$$\alpha_i = \max_{k \in S_i}(f_k + \min(\tilde{L}_{ik})). \tag{9}$$

Then, with the cost independence assumptions we have already made, we can rewrite equation (8) as

$$f_i = \sum_{z \geq \alpha_i} z \left(\prod_{k \in S_i} M_{ik}(z - f_k) - \prod_{k \in S_i} M^*_{ik}(z - f_k) \right). \tag{10}$$

A complete description of a practical procedure for computing the Fulkerson numbers f_i is given in Algorithm 1. At first blush this may not appear to be any simpler than before but, crucially, the number of operations required to compute each of the f_i is now $O(|\tilde{L}_{ik}| \cdot K)$, where K is the number of child tasks, rather than the first term being exponential in the second as before. Of course, it should be noted that this procedure is still more expensive than computing the u_i sequence.

Once all of the f_i have been computed, they can be taken as alternative task ranks in HEFT (or any listing heuristic). However, it should be emphasized here that although the f_i give tighter bounds on the critical path lengths of the stochastic DAG G_s there is absolutely no guarantee that this will lead to superior performance in the full heuristic. After all, G_s itself is only a model of how we expect the processor selection phase to proceed—one that we know for a fact is inaccurate since, for example, it implicitly assumes that all task and edge weights are independent. Indeed, without this independence assumption it is well-known that the relation $u_i \leq f_i$ does not necessarily hold even for G_s; Fulkerson himself presented examples [4]. Still, we think this is a reasonable enough basis for an alternative ranking method in HEFT, so we investigate its performance compared to the usual u_i ranks via numerical simulation in Section 4.

Two refinements of Fulkerson's method were proposed by Elmaghraby [3]. The first involves computing each of the f_i numbers in the aforementioned manner and then reversing the direction of the remaining subgraph in order to calculate an intermediate result which can be used to improve the quality of the bound. The second is a more general approach based on using two or more *point estimates* of e_i, rather than just f_i, a method that was later generalized by Robillard and Trahan [6]. In both cases Elmagharaby proved that the new number sequences achieve tighter bounds on e_i than the Fulkerson numbers f_i. However, small-scale experimentation suggested that the improvement of Elmaghraby's new bounds over Fulkerson's were typically minor compared to the improvement of the latter over the standard HEFT u_i sequence so we chose to only evaluate here whether tightening the bounds at all is useful.

Algorithm 1: Computing the Fulkerson numbers using Clingen's method.

for $i = n, \ldots, 1$ **do**
  $f_i = 0$, $\alpha_i = 0$, $Z_i = \{\}$
  for $k \in S_i$ **do**
    $\ell_m = \infty$
    for $\ell \in \tilde{L}_{ik}$ **do**
      $\ell_m \leftarrow \min(\ell_m, \ell)$
      if $f_k + \ell \notin Z_i$ **then**
        $Z_i \leftarrow Z_i \cup \{f_k + \ell\}$
      end
    end
    $\alpha_i \leftarrow \max(\alpha_i, f_k + \ell_m)$
  end
  for $z \in Z_i$ **do**
    if $z \geq \alpha_i$ **then**
      $g = 1$, $q = 1$
      for $k \in S_i$ **do**
        $g \leftarrow g \times M_{ik}(z - f_k)$
        $q \leftarrow q \times M^*_{ik}(z - f_k)$
      end
      $f_i \leftarrow f_i + z \times (g - q)$
    end
  end
end

3.2 Adjusting the pmfs

In some sense, the purpose of the node and edge pmfs m_i and m_{ik} is to simulate the dynamics of the processor selection phase of HEFT—i.e., $m_i(W_i^a)$ should represent the probability that task t_i is scheduled on processor p_a, and so on. In HEFT, tasks are assigned to the processor that is estimated to complete their execution at the earliest time and attempting to model this accurately beforehand can quickly get messy and expensive—especially given the interaction between the two phases of the algorithm. However, a sensible idea may be to simply *bias* the processor selection probabilities according to their relative power: if, say, a task is 10 times faster on one processor than another then it seems more likely it will be scheduled on the former than the latter, even once the effect of contention is taken into account. More precisely...

0

The expected values of the weights then become

$$\mathbb{E}[w_i] = \sum_{\ell \in L_i} \ell \hat{m}_i(\ell) = \frac{c_i n_c + g_i r_i n_g}{n_c + r_i n_g}, \tag{11}$$

$$\begin{aligned}\mathbb{E}[w_{ik}] &= \sum_{\ell \in L_{ik}} \ell \hat{m}_{ik}(\ell) \\ &= \frac{n_c(n_c-1)C_{ik}^c + n_c n_g (r_k C_{ik}^g + r_i G_{ik}^c) + n_g(n_g-1) r_i r_k G_{ik}^g}{(n_c + r_i n_g)(n_c + r_k n_g)},\end{aligned} \tag{12}$$

and these can be used with equation (5) to compute an alternative sequence of task ranks $\hat{u}_i$. Of course, this is slightly more computationally expensive than computing the standard u_i ranks but only by a constant factor. While there is no mathematically valid reason to suppose that the modified pmf is truly any more useful than the original, we evaluate its performance empirically using our simulation model at the end of this section.

4 Experimental comparison of rankings

Baseline comparison with random sort.

5 Processor selection

References

[1] Louis-Claude Canon and Emmanuel Jeannot. Correlation-aware heuristics for evaluating the distribution of the longest path length of a DAG with

random weights. *IEEE Transactions on Parallel and Distributed Systems*, 27 (11):3158–3171, 2016.

[2] C. T. Clingen. A modification of Fulkerson's PERT algorithm. *Operations Research*, 12(4):629–632, 1964.

[3] Salah E. Elmaghraby. On the expected duration of PERT type networks. *Management Science*, 13(5):299–306, 1967.

[4] D. R. Fulkerson. Expected critical path lengths in PERT networks. *Operations Research*, 10(6):808–817, 1962.

[5] Jane N. Hagstrom. Computational complexity of PERT problems. *Networks*, 18(2):139–147.

[6] Pierre Robillard and Michel Trahan. Technical note—expected completion time in PERT networks. *Operations Research*, 24(1):177–182, 1976.

[7] Richard M. Van Slyke. Letter to the editor—Monte Carlo methods and the PERT problem. *Operations Research*, 11(5):839–860, 1963.

[8] Henan Zhao and Rizos Sakellariou. An experimental investigation into the rank function of the Heterogeneous Earliest Finish Time scheduling algorithm. In *Euro-Par 2003 Parallel Processing*, Harald Kosch, László Böszörményi, and Hermann Hellwagner, editors, Berlin, Heidelberg, 2003, pages 189–194. Springer Berlin Heidelberg.