An Optimal Tile Allocation Algorithm for SET paper

We'll firstly define the concept of "Allocation Algorithm" and "Optimality" stated in the SET paper, then propose our algorithm (Alg. 1 below) and prove its optimality.

Definitions and The Allocation Algorithm

Firstly, we'll formally define the concept of a "tile allocation scheme" and the meaning of "optimality" stated in the SET paper.

Suppose we want to map N children onto C tiles, where each child has an NPT of $T_i, i = 1, 2, \ldots, N$. An allocation scheme is a function from $\{1, 2, \ldots, N\}$ to $\{1, 2, \ldots, C\}$, where $f(i)$ is the number of tiles allocated to child i, and f must satisfy $\sum_{i=1}^{N} f(i) \leq C$ to be valid. We can see that there exists a valid allocation scheme if and only if $C \geq N$.

For an allocation scheme f, the time of child i processing a batch is $\frac{T_i}{f(i)}$. Then the processing time of each batch is determined by the slowest child, which is $T_{proc}(f) = \max_{i=1,\ldots,n}(\frac{T_i}{f(i)})$, and minimizing unbalancement is equivalent to minimizing $T_{proc}(f)$. Thus an **optimal allocation scheme** f is a scheme which minimize $T_{proc}(f)$.

An **Allocation Algorithm** is an algorithm which inputs C, N and $T_1, \ldots, T_N$, and outputs a scheme f. Alg. 1 is the optimal Allocation Algorithm proposed by us, and the optimality of our algorithm can be stated as follows:

Theorem 1: For any input $C \geq N, \{T_1, \ldots, T_N\}$, Alg. 1 always outputs an optimize scheme f_{alg}.

In other words, for any scheme g, $T_{proc}(g) \geq T_{proc}(f_{alg})$

Both the algorithm (Alg. 1) and the proof starts at the next page.

The Allocation Algorithm

Algorithm 1: Optimal Tile Allocation Algorithm

// Here T_i is the NPT of the ith child
Input: Number of Tiles to be Allocated C, Number of Children n, List of NPT $T = \{T_i\}_{i=1}^{n}$
//Meaning of f: allocate $f(i)$ tiles for child i
Output: The Allocation Scheme $f : 1, 2, \dots, n \to \mathbb{Z}^+$
1 $sum\_T = \sum_{i=1}^{n} T_i$
2 // Calculate the largest #tiles below the ideal allocation of each child
3 $min\_alloc = \text{Dict}()$
4 **for** i **from** *1* **to** n **do**
5   // This is the number of tiles to be allocated ideally
6     $ideal\_alloc_i = C \times \frac{T_i}{sum\_T}$
7     $min\_alloc[i] = \text{floor}(ideal\_alloc_i)$
8   // This is the utilization when only allocating $min\_alloc[i]$ tiles
9     $min\_util[i] = \frac{min\_alloc[i]}{ideal\_alloc_i}$
10 **end**
11 $free\_tiles = C - \sum_{i=1}^{n} min\_alloc[i]$
12 **if** $free\_tiles == 0$ **then**
13   // The number of tiles in all children are ideal, return
14   **return** min_alloc
15 **end**
16 $f = \text{Dict}()$
17 **while** $free\_tiles > 0$ **do**
18   // Satisfy the child with the smallest min_util first
19     $j = \arg_{i \notin f} \min(min\_util[i])$
20     $f[j] = min\_alloc[j] + 1$
21     $free\_tiles -= 1$
22 **end**
23 // For the remaining children, recursively call this algorithm (Alg. 1, named ALLOC) to determine its allocation scheme
24 $remain\_tiles = C - \sum f[j]$
25 $remain\_n = n - \text{f.size}()$
26 $f' = \boldsymbol{ALLOC}(remain\_tiles, remain\_n, \{T_j \mid j \textbf{ not in } f\})$
27 // Merging the two schemes to get the whole allocation scheme
28 $f.merge(f')$
29 **return** f

Proof

We'll use induction on N to prove the correctness of Theorem 1.

When $N = 1$, at line 6 $ideal\_alloc_1$ will be set to C, then at line 11 $free\_tiles = 0$ and the returned f_{alg} (at line 14) will allocate all chips to the only child, which is obviously the optimal scheme.

Now suppose Theorem 1 holds for all $N \leq N_0$, we'll prove the returned f_{alg} is also optimal when $N = N_0$. Notice that since $\sum_{i=1}^{N_0} ideal\_alloc_i = C$ and $min\_alloc[i] \leq ideal\_alloc_i$, then at line 11 we have $free\_tiles \geq 0$.

If $free\_tiles = 0$, we must have $\forall i, min\_alloc[i] = ideal\_alloc_i$, then Algorithm 1 will return at line 14, and f_{alg} satisfies $\frac{T_i}{f_{alg}(i)} = \frac{\sum_{i=1}^{N_0} T_i}{C}$ which is a constant for different i. This means the allocation is fully balanced, and for any scheme g,

$$T_{proc}(g) = \max(\frac{T_i}{g(i)}) \geq \frac{\sum_{i=1}^{N_0} T_i}{\sum_{i=1}^{N_0} g(i)} \geq \frac{\sum_{i=1}^{N_0} T_i}{C} = T_{proc}(f_{alg}) \quad \text{,thus } f_{alg} \text{ is optimal here}$$

Now consider the case where $free\_tiles > 0$ at line 11. Then line 14 will be skipped and f_{alg} is returned in line 29. f_{alg} then is merged from two parts: f before line 28 and f' returned at line 26. Now denote $S_{N_0} = \{1, 2, \dots, N_0\}, S = \{i \in S_{N_0} \mid i \in f\}, S' = \{i \in S_{N_0} \mid i \in f'\}$, we have $S_{N_0} = S + S'$. Since the while loop at line 17 is executed at least once, S is not empty and $remain\_n < N_0$. Thus by the induction hypotheses f' is optimal on the subproblem $(remain\_tiles, remain\_n, \{T_j \mid j \in S'\}$ (since f' is returned by a call to Algorithm 1).

Denote $\alpha_i = C\frac{T_i}{\sum_{i=1}^{N_0} T_i}, R = \frac{\sum_{i=1}^{N_0} T_i}{C}$, for a scheme g we have $T_{proc}(g) = \max(\frac{\alpha_i R}{g(i)}) = \frac{R}{\min(\frac{g(i)}{\alpha_i})}$. Notice that we have $\sum_{i=1}^{N_0} \alpha_i = C \geq \sum_{i=1}^{N_0} g(i)$, so $\min(\frac{f(i)}{\alpha_i}) \leq 1$. Also α_i here is just $ideal\_alloc_i$ in the algorithm. Thus we have $f_{alg}(i) = \text{floor}(\alpha_i) + 1 > \alpha_i, \forall i \in S$, so the minimal $\min(\frac{f_{alg}(i)}{\alpha_i})$ is taken in S'.

Now suppose on the contrary, there exists a scheme g that satisfies $T_{proc}(g) < T_{proc}(f_{alg})$. Firstly we can't have $\forall i \in S, g(i) \geq f_{alg}(i)$ (**Statement 1**). If this statement is true, then $g(i) > \alpha_i, \forall i \in S$ and the minimal $\min(\frac{g(i)}{\alpha_i})$ is also taken in S'. However, $\sum_{i \in S'} g(i) \leq remain\_n$ and g restricted to S' is also a scheme on S', then by the optimality of f' we have $\min(\frac{f'(i)}{\alpha_i}) \geq \min(\frac{g(i)}{\alpha_i})$, which leads to a contradictory.

Thus there must exist $I \in S, s.t. g(i) < f_{alg}(i)$. Now construct a scheme f_1 as follows, $f_1(i) = f(i) = min\_alloc[i] + 1, \forall i \in S$ and $f_1(i) = min\_alloc[i], \forall i \in S'$. Notice that $free\_tiles$ are decremented by one in each while loop at line 17, so $|S| = C - \sum_{i=0}^{N_0} min\_alloc[i]$ and thus $\sum_{i=0}^{N_0} f_1(i) = C$. So f_1 is a valid scheme. Also, since $\forall i \in S, f_1(i) \geq f_{alg}(i)$, by Statement 1 $T_{proc}(f_1) \geq T_{proc}(f_{alg})$. Then we have $T_{proc}(f_1) > T_{proc}(g)$, which indicates $\min(\frac{f_1(i)}{\alpha_i}) < \min(\frac{g(i)}{\alpha_i}) \leq \frac{g(I)}{\alpha_I} \leq \frac{f(I)-1}{\alpha_I} = \frac{min\_util[I]}{1}$.

However, we also have $f_1(i) = f(i) > \alpha_i, \forall i \in S$, so $\exists j \in S', s.t. \min(\frac{f_1(i)}{\alpha_i}) = \frac{f_1(j)}{\alpha_j} = min\_util[j]$, which indicates $min\_util[j] < min\_util[I]$ for $j \in S'$ and $I \in S$. Which **contradicts with** the "argmin" at line 19 (since $I \in S$, I is selected as the argmin and added to f at one execution of the while loop at line 17, since $j \notin f$ is not the argmin, its min_util must not be smaller).

Thus Theorem 1 also holds for $N = N_0$. By induction we can prove the correctness of Theorem 1.