

# Modular Greedy Schedule Proof

evhiness

October 2016

0 is defined as the maximum finish value.

Jobs are given with integer start and end times or you're using a language that allows decimal modulus. If integer minutes are needed then simply let  $B = 2400$  and multiply each job by 100 i.e  $(18.30, 23.15) \rightarrow (1830, 2315)$ . Any number of finite decimal places can be added this way using big ints.

I will be using a modified version of the greedy choice algorithm from the book(iterative) as a subroutine:

```
// For simplicity, let job.s be its starting time, and job.f be its ending time.
// Finds the greedy scheduling rotated by a, mod B
SUBROUTINE GreedySch(S, a, B):
    n = S.length
    A = S[0]
    k = 0
    for m in range(1, n):
        if (s[m].s - a) mod B >= s[k].f - a mod B and s[k].f != 0:
            A.append(s[m])
            k = m
    return A

// Returns all elements that don't cross "rotated midnight"
SUBROUTINE noCross(S, p, B):
    n = S.length
    A = []
    // Assumes intervals of the form [a, b)
    for m in range(0, n):
        if ((S[m].s - p) mod B < (S[m].f - p) mod B) or (S[m].f == 0):
            A.append(S[m])
    return A
```

Required lemmas. Much longer than the actual algorithm which is on page 6.

Some notation: a "cross job" is a job that crosses some notion of "midnight" relative to some value  $0 \leq p < B$ . That is  $(a, b)$  such that  $(a - p) \geq (b - p) \pmod{B}$ ,  $b \neq 0$ . I will use both variants in the proofs below. In addition,

$(a, b) \equiv [a, b)$ , i.e. intervals are closed on the left and open on right. Thus jobs of the form  $(a, a) \equiv [a, 0) \cup [0, a)$  and thus take the entire 24 hours.

Lemma 1: A maximum solution contains at most one job that crosses "midnight".

Proof: As all intervals that cross "midnight" are of the form  $[a_c, 0) \cup [0, b_c)$ , they all, at least, contain 0 and thus cannot be scheduled together.

Lemma 2:

If  $(a, b)$  is able to be scheduled with  $(c, d)$  then  $((a-p) \pmod B, (b-p) \pmod B)$  is able to be scheduled with  $((c-p) \pmod B, (d-p) \pmod B)$ . I state this without proof as we're obviously just rotating each element by the same amount. Thus non-overlapping intervals remain non-overlapping.

Lemma 3:

There exists a  $0 \leq p < B$  such that the maximum solution, when rotated by  $p$ , mod  $B$  has no jobs that cross "midnight".

Proof:

Let  $Q$  be an optimal solution. Then by (1), we have that at most one of its elements crosses "midnight". If none cross, let  $p = 0$  and we are finished. Now assume  $(a_c, b_c)$  is the element that crosses. As  $Q$  is a solution, none of its elements overlap. That is to say  $(a, b)(c, d) \in Q \implies b \leq c$ .

Lay out of every element in  $Q$  that doesn't cross "midnight" in order of finish times i.e.  $(a_1, b_1)(a_2, b_2) \dots (a_n, b_n)$ . Then the cross term  $(a_c, b_c)$  must be such that  $a_c \geq b_n$  and  $b_c \leq a_1$  or else this would not be a valid solution. Rotate every element by  $a_c$ , mod  $B$ . This gives  $(0, b_c - a_c)(a_1 - a_c, b_1 - a_c) \dots (a_n - a_c, b_n - a_c) \pmod B$ . Note that  $b_1 < b_2 < b_3 \dots < b_n \leq a_c$ . Then  $\forall i \ a_i \leq b_i \leq a_c \implies a_i - a_c \leq b_i - a_c \leq 0 \implies 0 \leq a_i - a_c \leq b_i - a_c \pmod B$ . Thus this is a solution such that no elements cross "midnight".

Corollary of Lemma 3: For any schedule that contains the cross job  $(a_c, b_c)$ ,  $p = a_c$ . Therefore, we need only check  $p$  such that there is a cross job of the form  $(p, k) \in S$ . For example, if we have 3 cross jobs  $(a, b), (a, d), (w, v)$  then we need only check  $p = a$  and  $p = w$ .

Lemma 4:

If there exists a maximum solution, where  $(a_c, b_c)$  is its cross job, then it can be found using the greedy algorithm on  $(S - a_c) \pmod B$  directly.

Proof:

Assume a non-optimal solution is found by the greedy algorithm on the rotated space, then  $\exists (a_i, b_i), (v, w) \in S$  such that  $w > b_i$  but  $w - a_c < b_i - a_c \pmod B$ . Note that,  $w$  cannot be greater than  $a_c$  because  $w > v > a_c > b_c \implies w - a_c > v - a_c > 0 > b_c - a_c \implies b_c - a_c > w - a_c > v - a_c > 0 \pmod B$  and

$w > a_c > v > b_c \implies w - a_c > 0 > v - a_c > b_c - a_c \implies v_c - a_c > b_c - a_c > w - a_c > 0 \pmod{B}$  and therefore,  $(v, w)$  is either cross which, by Lemma 1, contradicts the fact that  $(a_c, b_c)$  is the cross job of the optimal schedule, or is wholly contained within  $(0, b_c - a_c)$  and thus would not be included in the schedule at all. So if  $w \leq a_c$ , we have  $b < w \leq a_c \implies b - a_c < w - a_c \leq 0 \implies w - a_c \leq b_i - a_c > 0 \pmod{B}$ . This is a contradiction of the fact that we chose  $w$  such that  $w - a_c < b - a_c$ . Therefore, no such  $(v, w)$  exists which implies that an optimal solution is found.

Hence, by lemma 3 and the corollary, if we check every rotation by  $p$  such that  $(p, v) \in S$ ,  $p > v$ , we can safely throw out cross terms for each iteration. And, by lemma 4, we can use the greedy algorithm directly on the rotated space which gives

$$OPT = \max_{(p,v) \in S, p > v} (greedySch(noCross(S, p, B), p, B))$$

$O(\max(Bn, n \log n))$

Thus this algorithm is  $O(n \log n)$  for constant  $B$  (modulus).

ALGORITHM ClockSchedule( $S, B$ ):

Let  $B = 24$  in this case

sort  $S$  in order of ascending finish times. (0 defined as maximum) //  $O(n \log n)$

$Q = \text{noCross}(S, 0, B)$  //  $O(n)$

$M = []$

$M.\text{append}(\text{greedySch}(Q, 0, B))$  //  $O(n)$

$V = S - Q$  //  $O(n)$

remove all duplicate start time elements from  $V$  (using a simple  $O(n \log n)$  algorithm)

// Note that we still check every element that crosses "midnight" relative to 0  
// but we only rotate to the starting hours that we need.

for  $q$  in  $V$ : //  $O(B)$

// Find all jobs that don't cross the starting time of  $q$

$W = \text{noCross}(S, q.s, B)$  //  $O(n)$

// Find the greedy schedule assuming the starting time of  $q$  is 0

$M.\text{append}(\text{greedySc}(W, q.s, B))$  //  $O(n)$

return (maximumBy (length)  $M$ ) //  $O(Bn)$