The state-flipping algorithm will not always find this configuration

For example, Let G be a graph consisting of a cycle of length four: there are nodes v_1, v_2, v_3, v_4 and edges (v_1, v_2), (v_2, v_3), (v_3, v_4), (v_4, v_1). Then if we start the state-flipping algorithm in a configuration where nodes v_1 and v_2 have state $+1$, and nodes v_3 and v_4 have state -1, then no improving move is possible.

[1] ex652.266.676

(a) Consider a path with three edges, and an execution of the greedy algorithm in which the middle edge is added first.

(b) Consider the k connected components $C_1, \ldots, C_k$ of $M \cup M'$ — each is path or a cycle. Label a component C_i by an ordered pair $(|M \cap C_i|, |M' \cap C_i|)$. Now, if some C has a label of the form $(0, j)$, then it follows that $j = 1$, and this is an edge of M' that can be added to M. Otherwise, the labels are $\{x_i, y_i\}$, where $x_i \geq 1$ and $y_i \leq x_i + 1$ for each i. But then $|M'| - |M| = \sum_i (y_i - x_i) \leq k$ while $|M| = \sum_i x_i \geq k$, so we have $|M| \geq |M'| - |M|$. Rearranging this last inequality, we get $|M'| \leq 2|M|$.

Another way to prove this is the following. Since no edge of M' can be added to M, each $e \in M$ shares an endpoint with some $e' \in M'$. (It may share an endpoint with two edges in M'; then pick one arbitrarily.) Make the edge $e' \in M$ "pay for" the edge $e \in M$. Now, each edge $e \in M$ has been paid for by some edge $e' \in M'$, but each $e' \in M'$ has only two endpoints and hence pays for at most two edges in M. It follows that M' contains at most twice as many edges as M.

(c) Let M' be a matching of maximum size, and let M be the matching obtained by the greedy algorithm when it finally terminates. Then since there is no edge from M' that can be added to M, it follows from (a) that $|M| \geq \frac{1}{2}|M'|$.

[1] ex406.701.840

(a) We can assume by symmetry that the first machine M_1 has the higher load. We need to prove that $T_1 \leq 2T_2$. Notice that our assumption that no single job takes more than half of the processing time implies the machine with higher load must have at least two jobs. Let job j be the smallest job on machine M_1. Clearly, $T_1 \geq 2t_j$. The local search algorithm terminated, so moving job j from machine M_1 to machine M_2 does not decrease the difference between the processing times. This implies that $t_j \geq T_1 - T_2$. We get that $T_1 \leq t_j + T_2 \leq \frac{1}{2}T_1 + T_2$, and multiplying by two we get that $T_1 \leq 2T_2$.

(b) We observed above that if a job j satisfies $t_j \geq |T_1 - T_2|$ it cannot no move. Further, the difference $|T_1 - T_2|$ decreases throughout the algorithm, so once this condition holds, job j will never move.

Now consider a job j, and we aim to prove that j will move at most once. Assume that jobs j starts on machine M_1. So the first time it moves it will move from machine M_1 to machine M_2. We always move the largest job, so at this point all remaining jobs on machine M_1 will have processing time at most t_j. Consider the sequence of consecutive moves all from machine M_1 to M_2, and let $t_{j'}$ be the last job that moves in this direction (possibly $j = j'$). At this time $T_2 \geq T_1$ and $T_2 - T_1 \leq t_{j'}$ as before moving job j' machine M_1 had more work. Now we have that $t_j \geq t_{j'} \geq |T_1 - T_2|$, so by the observation above job j will not move again.

(c) As an example of a bad local optimum let machine M_1 have two jobs with processing time 3 each. While machine M_2 has two jobs with processing time 2 each. Now the difference in loads is 2, no single job can move, but swapping a pair of jobs yields a solution with identical loads.

[1] ex511.777.306

(a) Consider a progress measure Φ defined as the sum of the squares of the loads on all machines. We claim that after an improving swap move, this quantity must strictly decrease. Indeed, suppose we execute an improving swap move on machines M_1 and M_2; suppose the loads on M_i and M_j before the swap are T_i and T_j respectively, and the loads after the swap are T_i' and T_j'. Then Φ decreases by $T_i^2 + T_j^2$ and increases by $(T_i')^2 + (T_j')^2$; since $T_i + T_j = T_i' + T_j'$, and $\max(T_i', T_j') < \max(T_i, T_j)$, it follows that the decrease is larger than the increase: in other words, Φ strictly decreases.

Since there are only a finite number of ways to partition jobs across machines, Φ can only decrease a finite number of times. This implies that the algorithm terminates.

(b) Consider the machine M_i with the maximum load at the end of the algorithm. If M_i contains a single job, then since this job has to go *somewhere* in any solution, the makespan of our solution is in fact at most the optimal makespan, so our solution is optimal.

Otherwise, M_i has at least two jobs on it. We now claim that the load on M_i is at most twice the load on any other machine M_j. Since in particular this will apply to the machine M_j of minimum load, it follows also that M_i is at most twice the average, $\frac{1}{m}\sum_r t_r$, from which it follows that we are within a factor of 2 of optimal.

So suppose $T_i > 2T_j$. Since M_i has at least two jobs, the lightest job r on M_i has size $t_r \leq \frac{1}{2}T_i$. So consider the swap move in which job r simply moves to M_i. If T_i' and T_j' are the loads after this move, we have $T_i' < T_i$ and $T_j' = T_j + t_r < \frac{1}{2}T_i + \frac{1}{2}T_i = T_i$. Thus $\max(T_i', T_j') < \max(T_i, T_j)$, so this is an improving swap move, contradicting the termination of the algorithm.

[1] ex798.837.852