Appendix

A Implementation Details

We use the publicly available source code for each of the baseline methods and we use their default parameter settings in our experiments. For our method, we adopt the GCN described in Equation (2) to implement the assignment network g_θ, and we use $\xi = 5$ and $K = 2$ by default if not specified otherwise. For the training algorithm (Algorithm 1) of our method, we use a batch size of 32, use the learning rate $\eta_1 = 0.01$, set the learning rate η_2 to be 0.2 on the Facebook dataset (illustrated later) and 0.005 on the other datasets, set λ to be 0.2 on the Facebook dataset and 0.1 on the other datasets . For all the compared methods, we use the same victim model as [37, 38] when training the corresponding attacks of each method. All experiments are conducted on a server with an NVIDIA RTX 3090 GPU, 64 GB main memory and an Intell(R) Core(TM) i9-10900K CPU @ 3.70GHZ. Each experiment is independently repeated 5 times to report the average performance.

Table 7: Fooling ratio vs. K when $\xi = 5$.

K	Cora	Citeseer	Facebook	Wiki
1	81.36%	79.14%	8.33%	83.43%
2	94.68%	92.77%	16.38%	93.56%
3	95.23%	92.77%	16.38%	93.56%
4	95.23%	92.77%	16.38%	93.87%

B The Effect of K on Fooling Ratio

We show in Table 7 the effect of K on the fooling ratio (FR) of the proposed MFAN attack when $\xi = 5$. On all the datasets, FR significantly increases when K increases from 1 to 2. This is because, when $K = 1$, MFAN uses a single set of anchor nodes, which is not enough to successfully attack all the target nodes. When $K = 2$, MFAN uses two sets of anchor nodes, each of which is specialized to successfully attack a different set of target nodes. Moreover, MFAN also ensures the success rate of each attack by using the assignment network to choose the best set of anchor nodes for each attack. In this way, MFAN is essentially performing "divide and conquer", and the final set of target nodes successfully attacked by MFAN is close to the union of the target nodes that are successfully attacked by each of the K sets of anchor nodes. When K becomes larger than 2, FR converges quickly due to the well-known "diminishing marginal effect", that is, most of the target nodes successfully attacked by the 3rd set of anchor nodes

Table 8: Average attacking time in milliseconds.

Methods	Cora	Citeseer	Facebook	Wiki
GUA	146	233	-	112
GUAP	176	259	-	139
PGD	≈ 0	≈ 0	≈ 0	≈ 0
DICE	≈ 0	≈ 0	≈ 0	≈ 0
Meta-Self	≈ 0	≈ 0	≈ 0	≈ 0
FGA	468,370	1,008,467	1,539,644	357,708
MFAN	154	240	375	131

are already successfully attacked by the previous two sets of anchor nodes, thus using the 3rd set of anchor nodes does not improve FR very much. As a result, using two sets of anchor nodes is good enough for MFAN to achieve outstanding FR. Since the BFA of MFAN is $\delta = K * \xi$, we set $K = 2$ by default for the rest experiments to save our BFA while achieving outstanding FR.

C Attacking Time

Table 8 shows the average attacking time for each compared method, that is, the average time cost to attack each target node in the testing dataset. For each method, we use a total number of 10 controlled nodes, which means setting $\xi = 10$ for FGA and setting $\delta = 10$ for the other methods. As shown in Table 8, FGA costs the largest attacking time because it requires training a unique set of controlled nodes from scratch for each new attack, such training cannot be done offline and it can only begin when FGA starts to attack a new target node, therefore the attacking time of FGA is the same as its training time. Comparably, the anchor nodes methods are achieving a much faster attacking time than FGA, because they only require to flip a small number of edges when attacking a new target node, and the anchor nodes are trained offline before launching any attack. The attacking time of MFAN is slightly larger than GUA because MFAN needs a forward pass of the assignment network to choose the best set of anchor nodes. GUAP costs slightly more attacking time than MFAN because it needs to inject new anchor nodes into the graph G. The global structural attacks, such as PGD, DICE and Meta-Self, do an offline training to perturb a large number of edges once and for all (i.e., for only a single time). Such an attack format does not need to do anything when attacking new target nodes, thus their attacking time is approximately zero. However, their attack effectiveness under a limited budget of controlled nodes is also significantly restricted by their attack format.

D Training Time

In this section, we first analyze the time complexity for the training of all the methods. Then, we evaluate and compare the actual training times between our method and the baselines.

Table 9: Training time complexity analysis

Methods	Time Complexity
GUA	$O(max\_epoch \cdot N_T \cdot max\_iter)$
GUAP	$O(max\_epoch \cdot N_T \cdot max\_iter)$
PGD	$O(T)$
DICE	$O(1)$
Meta-Self	$O(\delta)$
FGA	$O(N\xi)$
MFAN	$O(max\_epoch \cdot N_T K)$

Table 10: Average training time in milliseconds.

Methods	Cora	Citeseer	Facebook	Wiki
GUA	13,084,775	24,971,433	-	9,508,160
GUAP	9,422,330	18,956,390	-	7,440,657
PGD	14,257	19,360	30,404	9,617
DICE	165	193	412	140
Meta-Self	6,798	7,217	8,145	6,211
FGA	468,370	1,008,467	1,539,644	357,708
MFAN	588,663	1,021,496	1,845,367	450,552

The key determinant of training time is the execution of forward and backward passes through the victim model GCN. Table 9 summarizes the time complexity of each method based on the number of GCN passes. Denote by N_T the number of nodes in the training set. Our method, following the optimization objective in Equation (9), requires $N_T K$ GCN calls per epoch, resulting in a complexity of $O(max\_epoch \cdot N_T K)$. GUA and GUAP, as anchor nodes attacks, also perform GCN calls for each target node every epoch but constantly adjust perturbation vectors to check misclassification results, with the number of checks bounded by max_iter, leading to a complexity of $O(max\_epoch \cdot N_T \cdot max\_iter)$. PGD and Meta-Self, as global methods, update the entire graph structure, producing a single perturbed adjacency matrix per epoch. For PGD, the time complexity is solely bounded by the number of iterations $O(T)$, whereas Meta-Self performs δ times of perturbations, leading to $O(\delta)$ time complexity. DICE, which modifies edges entirely on topological structure, is not gradient-based and does not involve any operation with GCN, resulting in constant time complexity $O(1)$ in terms of GCN passes. FGA, being a targeted attacking method, modifies one edge per target node based on the largest gradient for ξ iterations, making its time complexity $O(N\xi)$ in order to attack all target nodes.

We measured the actual training time attacking all target nodes, as detailed in Table 10, setting $\xi = 10$ for FGA and $\delta = 10$ for the other methods.

Given that max_iter significantly exceeds our K, our method is more efficient, with training time in minutes, contrasting with the hours required by GUA

and GUAP. The global methods PGD, Meta-Self, and DICE, which do not require per-node GCN passes, run faster than ours by a factor of N_t, completing in seconds or milliseconds. FGA, to generate attacks for all target nodes, requires $O(N\xi)$ GCN calls in total and exhibits a similar training time magnitude to ours.