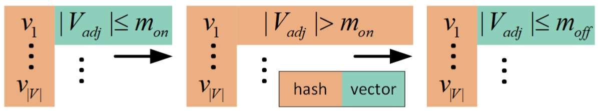

Figure 1: A non-conventional adjacency list. We store vertices $v_1, \cdots, v_{|V|}$ in a hash. Let V_{adj} be the set of adjacent vertices of v_1. m_{on}, m_{off} are two values, and $m_{on} \ge m_{off}$. Initially, if $|V_{adj}| \le m_{on}$, we store V_{adj} in a vector. After adding edges, if $|V_{adj}| > m_{on}$, we store V_{adj} in a hash. After removing edges, if $|V_{adj}| \le m_{off}$, we store V_{adj} in a vector.

graph_hash_of_mixed_weighted

"graph_hash_of_mixed_weighted" is a non-conventional adjacency list of mixed hashes and vectors to store an undirected and weighted graph G (see Figure 1). In this adjacency list, we use a hash to store all vertices. By doing this, we can access every vertex within $O(1)$ time. We set two constant values m_{on}, m_{off} such that $m_{on} \ge m_{off}$. Let V_{adj} be the set of adjacent vertices of vertex v_1. Initially, if $|V_{adj}| \le m_{on}$, we store V_{adj} in a vector. After adding edges, if $|V_{adj}| > m_{on}$, we store V_{adj} in a hash. After removing edges, if $|V_{adj}| \le m_{off}$, we store V_{adj} in a vector again. The purpose of using vectors and hashes to store small and large sets of adjacent vertices respectively is to employ both the small memory consumption of vectors and the small time complexities of hashes. The space time complexity of this adjacency list is $O(|V| + |E|)$. Since the size of a vector in this adjacency list is constrained by m_{on}, the time complexity of adding or removing an edge is $O(1)$.