Comparing performance of different hash algorithms

Small dict	**bins**	**occupancy**	**used_bin_fraction**	**max_entries**
naive_hash	256	0.390625	0.304688	3
bernstein_hash	256	0.390625	0.320312	3
FNV_hash	256	0.390625	0.335938	4

Medium dict	**bins**	**occupancy**	**used_bin_fraction**	**max_entries**
naive_hash	4096	0.488281	0.181885	12
bernstein_hash	4096	0.488281	0.388184	4
FNV_hash	4096	0.488281	0.385986	4

Large dict	**bins**	**occupancy**	**used_bin_fraction**	**max_entries**
naive_hash	131072	0.633438	0.012505	283
bernstein_hash	131072	0.633438	0.469856	7
FNV_hash	131072	0.633438	0.470131	6

It is clear to see that the naive_hash function has performance deficiency compared to the other 2 algorithms. This is clear when observing its low used_bin_fraction and high max_entries, both in the medium and large dictionary. Hence, naive_hash is clearly a bad hashing function to store english words.

The reason relies on how the algorithm is designed. If we look at its code, we can see that what the algorithm does is sum up all the integer values of each of the chars of the word being hashed, take that sum - lets call it s - and then calculate its mod number_of_bins value. Now, english words are quite limited in their number of characters, so s is usually bounded in its value. Let's call s_m the maximum value of s over all the words of the dictionary. Once number_of_bins starts growing and is over the value of s_m, none of the buckets with index between s_m and number_of_bins will be used. This is why its used_bin_fraction is so low, and hence max_entries tend to increase.

To give an example, in the case of medium_dictionary, s_m is 2083, corresponding to the word 'incontrovertibility'. Due to the amount of words in medium_dictionary, in order to keep a load_factor<0.75, the array has 4096. Hence, no word of the dictionary is matched to the binds from index 2084 to 4095.