There are several hashing and collision resolution techniques, each with their own advantages and limitations. They include but not limited to:

Linear Probing: Linear probing is a collision resolution technique that involves checking the next slot in the hash table when collisions occurs. If that slot is already occupied the algorithm moves on to the next slot and so on until when the next slot is found. The major advantage of linear probing is that it is simple and requires minimal memory usage. However it suffers from clustering; which is the tendency of items to accumulate in the same vicinity. As a result, it can lead to longer search times and slower insertion times as the table fills up.

Quadratic Probing: Quadratic probing is similar to linear probing, but instead of checking the next slot when collision occurs, it checks slots that are quadratic distance away. The advantage of quadratic probing is that it reduces the clustering effect of linear probing. However it still suffers from clustering and performance can still degrade significantly overhead in accessing and maintaining the linked list.

Chaining: Chaining involves using linked list within each slot hash table to store items that hash to the same value. The advantage of chaining is that it allows for an unlimited number of items to be stored in each slot, reducing the chance of collision. However it requires additional memory to store the linked list. And there can be significant overhead in accessing and maintaining the linked lists.

Multiplication hashing: Multiplication hashing involves multiplying the key by a constant factor and then using the fractional part of the result to compute the hash value. This technique is known to produce good distribution of hash values and can reduce the clustering effect of linear probing. However, it requires floating-point arithmetic, which can be slower than integer arithmetic on some systems.

ii) to address some of the problems encountered with these techniques, one approach is to use combination of techniques. For example a hash table can be implemented using chaining, with either linear or quadratic probing used to solve collisions within the each slot. Alternative techniques such as cuckoo hashing or Robin Hood hashing can be used, which involves moving items around hash table to resolve collisions and maintain a more balanced distribution of items. Additionally, resizing the hash table when it becomes too full can also help to reduce the clustering effect and maintain performance.

iii) Load factor is an important factor in determining the performance of the hash table. It represent the ration of the number of elements stored in the hash table. A higher load factor indicates that the hash table is more full and can lead to a higher probability of collisions, which can lower the performance of the hash table. Basically a load factor of around 0.7 is optimal for the hash table, as it provides good balance between space usage and collision avoidance. If the load factor is too low, the hash table will be less efficient in terms of space usage. While load factor that is too high can result to many collisions, leading to slower performance.

When the load factor exceeds a certain threshold. It may be necessary to resize the hash table to maintain performance. Resizing involves creating a new, larger hash table and rehashing all the elements from the old table to the new table. This can be an expensive operation, so it is generally best to choose an appropriate initial size for the hash table based on the expected number of elements and the desired load factor, in order to minimize the need of resizing.

iv) When deleting items from a table, it is imporatant to consider the potential ramifications as it can affect the structure and performance of the hash table.

One issue that arise when deleting items from a hash table is that it can create gaps in the table, which can lead to inefficiencies in searching for other items. In the case of open addressing resolutions techniques such as linear probing or quadratic probing, deleting an item can leave being empty cells that need to be skipped over during searches, potentially increasing the time complexity of operations.

Another issue to consider is the impact of deletion on the load factor of the hash table. If a large number of items are deleted, the load factor may decrease significantly leading to underutilization of the hash table.

In bioinformatics hash tables are commonly used in the application such as sequence alignment and database indexing. In these applications, it is often necessary to delete items from the hash table as new data is added or existing data modified for example, in a genome assembly application, a hash table might be used to store k-mers from DNA sequences. As new sequences are added to the assembly, some k-mers may become redundant and can be deleted from the the hash table to save memory and improve performance

Overall, while deleting items from a hash table can be useful in managing memory and improving performance, it is important to carefully consider the potential ramifications and implement appropriate strategies for managing gaps and load factor.