2. GA-based TSP TSP is known as an NP-hard program that causes a computational explosion. For instance, finding the shortest route through 36 cities needs to examine 36! (= 36 × 35 × … × 1) combinations. GA is quite effective to reduce TSP's computation time while reaching a semi-optimal trip (but no the shortest path). Consider a travel through 36 cities, each named with one of the 36 characters such as A~Z and 0~9.
ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789 is one possible trip. In GA, this string and each city in it can be considered as a chromosome and a gene respectively. We will first generate 50,000 different trips or chromosomes, and then repeat 150 iterations or so-called generations, each including: (1) evaluate( ): evaluates the distance of each trip and sorts out all the trips in the shortest-first order. Memorize the current shortest trip as a tentative answer if it is shorter the previous. (2) select( ): selects the shortest 25,000 trips as parents. (3) crossover( ): generates 25,000 off-springs from the parents. More specifically, we spawn a pair of child[i] and [i+1] from parent[i] and [i+1]. (4) mutate( ): randomly chooses two distinct cities (or genes) in each trip (or chromosome) with a given probability, and swaps them. (5) populate( ): populate the next generation by replace the bottom 25,000 trips with the newly generated 25,000 off-springs.
3. Crossover Algorithm The key to GA-based TSP is to design a suitable crossover algorithm. A typical crossover generates child[i] by combining the first half of parent[i]'s genes and the last half of parent[i+1]'s genes, whereas gives child[i+1] the last half of parent[i]'s genes and the first half of parent[i+1]'s genes. However, this crossover does not work in TSP. For example in a TSP program for visiting only eight cities, consider two parents: parent[i] = ABCDEFGH parent[i+1] = HGABFECD Their children will be:: child[i] = ABCDFECD child[i+1]=HGABEFGH
Child[i] and [i+1] will end up with revisiting CD and GH respectively. To address this problem, we will use a greedy crossover algorithm: We select the first city of parent[i], compares the cities leaving that city in parent[i] and [i+1], and chooses the closer one to extend child[i]'s trip. If one city has already appeared in the trip, we choose the other city. If both cities have already appeared, we randomly select a non-selected city. Thereafter, we generate child[i+1]'s trip as a complement of child[i].