Inversion mutation is a mutation operator for combinatorial search domains. It is based on the random change of edges. A famous example for a combinatorial problem is the traveling salesman problem (TSP). Inversion mutation is a typical genetic operator for combinatorial problems like the TSP. It reverses the tour between two randomly chosen cities. Here, we propose a self-adaptive variant of inversion mutation. Self-adaptation is a successful control technique for the mutation strength, see chapter 3. Up to now, the concept of self-adaptation has not been applied to inversion mutation. As the number of successive inversion mutation operator applications can be seen as the mutation strength, we propose to control this parameter self-adaptively. In this chapter we introduce a self-adaptive variant of inversion mutation (SA-INV). We prove the convergence of inversion mutation and its self-adaptive variant and show experimentally that SA-INV speeds up the evolutionary process, in particular at the beginning of the search. Later we encounter the strategy bound problem and introduce a heuristic to overcome it.

5.1 Introduction

At first, we introduce the TSP and give an overview of evolutionary combinatorial optimization and self-adaptation for discrete strategy variables.

5.1.1 The Traveling Salesman Problem

The TSP is the problem to minimize the length of a salesman’s tour visiting every city of a given set only once and then returning back to the point where he started. It is defined as follows.

Definition 5.1 (Traveling Salesman Problem)

Let $C$ be a set of $N$ cities with distances $d(c_i, c_j) \in \mathbb{R}$ for each pair of cities $c_i, c_j \in C$. An optimal solution of the TSP is the shortest tour $\pi^*$ of $C$, i.e. a permutation $\pi : [1, \ldots, N] \leftrightarrow [1, \ldots, N]$ with minimum length $l = \sum_{i=1}^{N-1} d(c_{\pi(i)}, c_{\pi(i+1)}) + d(c_{\pi(N)}, c_{\pi(1)})$. 
TSP is a very famous combinatorial optimization problem with dozens of solution approaches. The exact solution of bigger TSP instances is impractical since the problem is NP-complete [45]. With dynamic programming a solution can be computed in time $O(n^2 \cdot 2^n)$, which is still impractical. Other exact methods to solve the TSP are branch-and-bound and linear programming approaches. Various heuristics exist to solve the TSP reaching from nearest-neighbor strategies to the 2-opt heuristic. 2-opt is a famous heuristic replacing two longer random edges by two shorter ones [26]. The concept is based on the idea of inversion mutation: we select a subtour, reverse it and accept an improvement. An approach similar to simulated annealing accepts improvements with a certain probability, e.g. recently theoretically analyzed by Meer [89]. He constructs a TSP instance for which simulated annealing outperforms a metropolis algorithm with a fixed temperature using the 2-opt heuristic.

5.1.2 Evolutionary Combinatorial Optimization

Combinatorial optimization problems are believed not to be efficiently solvable, most problems are NP-hard. Examples for combinatorial problems are the knapsack problem or the TSP. For the latter the solution of trying all permutations is impractical as the number of possible solutions is $n!$. In the traditional representation the queue of integers represents the order of the tour in which the cities have to be visited. To avoid multiple nodes, which would happen after the application of n-point crossover, partially mapped crossover (PMX) is used, see chapter 6. The random keys approach avoids infeasible solutions [10], [146]: Each city is assigned with a random number $x \in [0, 1)$. The chromosome is produced by visiting the nodes in ascending order. As this order is always well-defined as long as no equal numbers exist, crossover operators like N-point crossover can be applied. Other evolutionary approaches are successful like memetic algorithms, which combine EAs with local optimization methods, e.g. 2-opt.

5.1.3 Self-Adaptation for Discrete Strategy Variables

Not many EAs with discrete strategy variables exist. In chapter 6 we introduce self-adaptive crossover. Self-adaptive crossover controls the position of crossover points which are represented by integers [74]. Punctuated crossover by Schaffer and Morishima [128] makes use of a bit string of discrete strategy variables which represent location and number of crossover points. In section 4.6 we examined the self-adaptive selection of mutation operators for ES. Spears [147] introduced a similar approach for the selection of crossover operators. Bäck [7], Smith [142], Fogarty [144] and Stone [148] introduced self-adaptive approaches for mutation rates.

---

1 NP-hard (nondeterministic polynomial-time hard): a problem H is NP-hard iff there is an NP-complete problem L that is polynomial time Turing-reducible to H, i.e. $L \leq_T H$. 

---