Backtracking

General recursive pseudocode of Backtracking (based on lecture from previous years):

```
bKt(s)   // s = partial solution
{
    if(s is a complete solution)
        return s
    else {
      for s' in successors(s) {
          if(s' is a viable solution) {
              sol = bKt(s')
              if(sol!=0)
                  return sol
          }
      }
      return 0
    }
}

bKt( initial solution)
// bKt func. currently returns { a solution s
                               { 0, if no solution is found
```

A solution is usually represented as a vector/matrix.

Partial solution = a solution under construction (usually a "prefix" of a solution)

Complete solution = final solution (there can be multiple complete solutions)

Viable solution = a solution that has chances to become a complete solution (no restrictions are currently unsatisfied)

Successors of a partial solution = solutions that can be built by making one step starting from the partial solution (e.g., adding one more element)

Practical tips:

- Sometimes, when we come back from a recursive call, we need to "clean" a position in the vector/matrix representing the solution
- To not stop after finding one solution, eliminate the returns from the pseudocode

Used in the context of optimization problems.

At each step, we retain a "bestSoFar" = current best value of a solution of the problem

If the partial solution, continued in the best possible way cannot become a better solution than "bestSoFar", then quit the search on that branch (prunning).

For example, in case of maximization problems (knapspack problem):

If partialProfit + maxRest <= bestSoFar then prune partial solution

partialProfit = current profit of the partial solution sol[0..i-1]
maxRest = maximum profit that could be obtained by selecting all elements not taken into account yet
(= v[i]+v[i+1]+…+v[n-1])