Solution 1

```go
// Copyright © 2020 AlgoExpert, LLC. All rights reserved.

package main

func KnapsackProblem(items [][]int, capacity int) []interface{} {
  values := make([][]int, len(items)+1)
  for i := range values {
    values[i] = make([]int, capacity+1)
  }
  for i := 1; i < len(items)+1; i++ {
    currentValue := items[i-1][0]
    currentWeight := items[i-1][1]
    for c := 0; c < capacity+1; c++ {
      if currentWeight > c {
        values[i][c] = values[i-1][c]
      } else {
        values[i][c] = max(values[i-1][c], values[i-1][c-currentWeight]+currentValue)
      }
    }
  }

  value := values[len(items)][capacity]
  sequence := getKnapsackItems(values, items)
  return []interface{}{value, sequence}
}

func getKnapsackItems(values [][]int, items [][]int) []int {
  sequence := []int{}
  i, c := len(values)-1, len(values[0])-1
  for i > 0 {
    if values[i][c] == values[i-1][c] {
      i--
    } else {
      sequence = append(sequence, i-1)
      c -= items[i-1][1]
      i--
    }
    if c == 0 {
      break
    }
  }
  reverse(sequence)
  return sequence
}

func reverse(numbers []int) {
  for i, j := 0, len(numbers)-1; i < j; i, j = i+1, j-1 {
    numbers[i], numbers[j] = numbers[j], numbers[i]
  }
}

func max(arg int, rest ...int) int {
  curr := arg
  for _, num := range rest {
    if curr < num {
      curr = num
    }
  }
  return curr
}

```