Algorithmen, KI und Data Science 1 (AKIDS 1): **Metaheuristic & Constrained Satisfaction**

Herbst- / Wintersemester 2023/24

Prof. Dr. Goran Glavaš

Benedikt Ebing

Fabian David Schmidt

Metaheuristic Search

Exercise 1.1

How do state space search problems differ from discrete constraint optimization problems?

Recap: What is state space search?

Recap State Space Search

- ("Small") Set of start states
- Transitions between states resulting in large number of possible successor states
- Finding a goal state that meets a particular criterion "usually" with minimal cost/maximal gain
- Defined as:

$$sss = (s_0, succ, goal), \text{where}$$
$$initial\ state\ s_o \in \boldsymbol{S},$$
$$succ: \boldsymbol{S} \rightarrow f(\boldsymbol{S}),$$
$$goal: \boldsymbol{S} \rightarrow \{True, False\}$$

Recap: What are discrete optimization problems?

Recap Discrete Optimization Problem

- Large set of states, each is a possible solution
- Function that assigns a quality to each state
- Set of constraints that specifies which states are valid solutions
- Goal to find the optimal state (i.e., state with highest quality)

How do state space search problems differ from discrete constraint optimization problems?

Exercise 1.1 – Solution

- Starting point: set of initial states (small subset of state space) vs. whole state space
- Next state: compute successor states based on allowed transitions vs compute neighborhood states (usually based on distance)
- Ending point: goal state based on goal test (small subset of state space) vs. no explicit goal test, try to find state with maximum/minimum value of objective function f

Exercise 1.1 – Solution

- Problem to solve: How to get to the goal state with minimal cost/maximal gain vs. find the state with minimal/maximal value
- Guidance: Specific heuristics (estimate distance to goal state) to trim the number of paths to explore vs. problem agnostic metaheuristics to find next best solution in the neighborhood

Exercise 1.2

Define the 0-1 knapsack problem as discrete constrained optimization problem. The 0-1 knapsack problem is defined as follows: given a set of N items, each with a weight w and a value v, determine the subset of items to choose such that the total weight is less than or equal to a given limit W and the total value is as large as possible.

Exercise 1.2 – Solution

- $X = (x_1, x_2, \ldots, x_N)$, where N is the total number of items
- $D_1 = D_2 = \cdots = D_N \in \{0,1\}$ because each element is either included in the knapsack or not
- $f(X) = \sum_{i=1}^{N} v_i x_i$, where v_i refers to the value of item x_i
- Constraint: $\sum_{i=1}^{N} w_i x_i \leq W$, where w_i refers to the weight of item x_i

Exercise 1.3

Recap: What is a single point search algorithm?

Recap Single Point Search Algorithm

- Examine one state (candidate solution) at a time
- Usually choose next candidate solution from local neighborhood of current solution

Recap: How does simple descent work?

Recap Simple Descent

- Select any state s' from the neighborhood that is better than current state s

```
simple_descent(s, N)
 while True
   better = False
   for s' in  N(s)
      if f(s') < f(s)
        s = s'
        better = True
        break
   if not better
     break
 return s
```

How does simulated annealing escape local optima?

Exercise 1.3 – Solution

- Allow suboptimal moves with decreasing probability through the search

```
simulated_ annealing(s0, N , T, end)
  iter = 0
  while not end(T, iter)
    iter = iter + 1
    s' = randomly select from N(s)
    if f(s') < f(s)
      s = s'
    else
      p = exp(-(f(s') - f(s))/T)
      p' = random(0, 1)
      if p' < p
        s = s'
  return s
```

Exercise 1.4

Recap: What is a population-based search algorithm?

Recap Single Point Search Algorithm

- Examine more than one state (candidate solution) at a time
- Between iterations, the population is partially or completely replaced

Recap: How do genetic algorithms work?

Recap Genetic Algorithm

- Examine more than one state (candidate solution) at a time
- Between iterations, the population is partially or completely replaced

```
genetic_algorithm(S, end)
  p = create_init_population(S)
  iter = 0
  evaluate(p)
  while not end(p, iter)
    iter = iter + 1
    p' = recombine(p)
    mutate(p')
    evaluate(p')
    p = select(p U p')

  return p
```

How do genetic algorithms escape local optima?

Exercise 1.4 – Solution

- Genetic algorithm uses mutation (random change of values in the chromosome) to escape local optima (e.g., element swap or element change)

Constrained Satisfaction Problems

Recap: What is a constrained satisfaction problem?

Recap – Constrained Satisfaction Problems

- State $X = (x_1, x_2, \dots, x_n)$ that can be factored in n variables
- Corresponding domains $D_1, D_2, \dots, D_n$
- State satisfies (set of) constraints C
- Find variable assignment that satisfies the constraints
- All solutions equally good
- Partial solutions that violate the constraints cannot be part of the solution.

Exercise 1.1

How do state space search problems differ from constrained satisfaction problems?

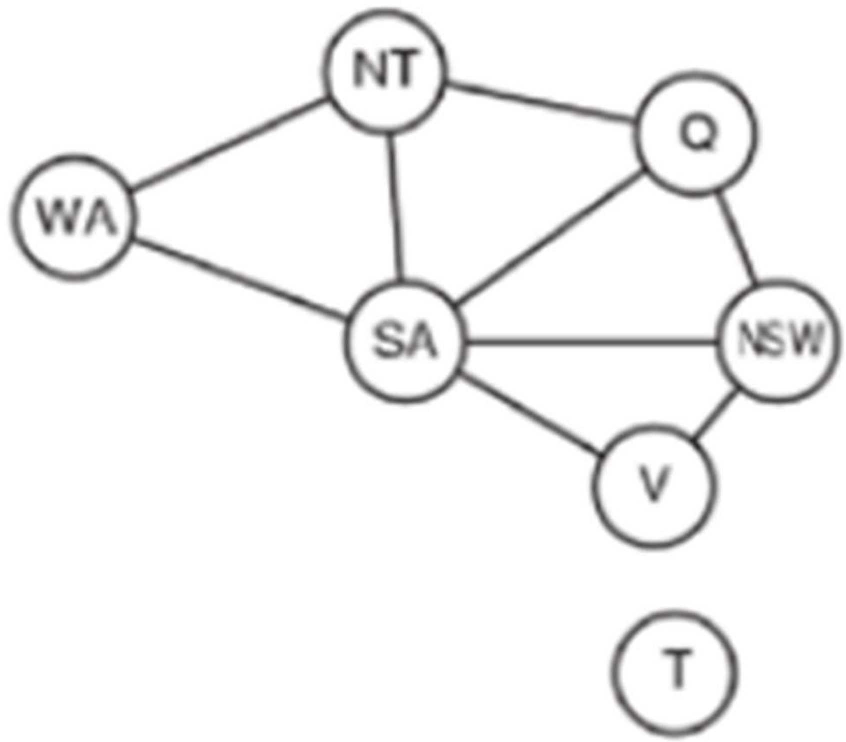
Exercise 1.1 – Solution

- Type of problem: Optimal path problems vs. finding any solution (all equally good)
- Type of states: Complex, non-factorable states vs. factorable states
- Type of transitions: Explicit state transitions defined by the nature of the problem vs. transition from k+1 assigned variables

Exercise 1.2

How many solutions does the map coloring problem from lecture 18 have? How many if two colors are allowed?

Exercise 1.2 – Solution



Exercise 1.2 – Solution

- Mainland: $3 \times 2 = 6$ solutions
- Tasmania not connected to the mainland: 3 solutions
- Total: $3 \times 6 = 18$

- With two colors no solution exists

Exercise 1.3

Recap: How does the backtracking algorithm work?

Recap: Backtracking algorithm

```
backtracking_search(csp)
   return backtrack({}, csp)


backtrack(s, csp)
  if complete(s) return s
  v = select_unassigned_var(csp.vars)

  for val in order-values(v, s, csp)
    if not csp.violates(s U (v, val))
      csp.vars[v] = val
      res = backtrack(s U (v, val), csp)
      if res ≠ null
         return res
  csp.vars[v] = null
  return null
```

Recap: What is inference or constraint propagation?

Recap: Inference (constrained propagation)

```
backtracking_search(csp)
   return backtrack({}, csp)

backtrack(s, csp)
  if complete(s) return s
  v = select_unassigned_var(csp.vars)
  for val in order_values(v, s, csp)
    if not csp.violates(s U (v, val))
      csp.vars[v] = val
      infs = csp.inference(v, val)
      if infs = null # violation
        continue
      res = backtrack(s U (v, val), csp)
      if res ≠ null
         return res
      csp.remove(infs)
  csp.vars[v] = null
  return null
```

Recap: In which methods are the MRV and LCV heuristic applied?

Recap: MRV and LCV

```
backtracking_search(csp)
   return backtrack({}, csp)

backtrack(s, csp)
  if complete(s) return s
  v = select_unassigned_var(csp.vars)
  for val in order_values(v, s, csp)
    if not csp.violates(s U (v, val))
      csp.vars[v] = val
      infs = csp.inference(v, val)
      if infs = null # failure
        continue
      res = backtrack(s U (v, val), csp)
      if res ≠ null
         return res
      csp.remove(infs)
  csp.vars[v] = null
  return null
```

Explain why choosing the variable that is most constrained, but the value that is least constrained is considered a good heuristic?

Exercise 1.2 – Solution

- MRV: fail early ➔ prune large parts of the search tree
- LCV: higher chances of avoiding conflicts in upcoming steps, increasing the chances of finding an actual solution

Exercise 1.4

Consider the problem of placing k knights on an $n \times n$ chessboard such that no two knights are attacking each other, where k is given and $k \leq n^2$. Choose a CSP formulation giving your (a) variables, (b) the domain of each variable, and (c) the constraints

Exercise 1.4 – Solution

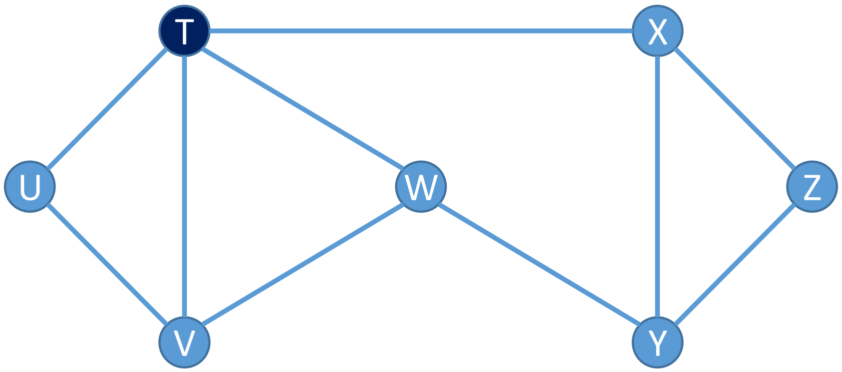
- Variables: $X = (x_1, x_2, \dots, x_k)$, where k is the number of knights placed on the chess board and x_i represents the position of each knight.
- Domain: $x_i \in \{(a,1),(a,2),\dots,(h,8)\}$, where each tuple is a position on the chess board
- Constraints: Two knights cannot share the same position AND a move from any knight does not result in two knights sharing the same position
- Alternative: Variables are the positions of the chess board and domains are {0,1} ➔ Constraints need to be changed accordingly

Exercise 1.5

Fill out the table below by applying the backtracking algorithm with *inference*, *degree heuristic* and *least constraining value* heuristic to the coloring problem. In case the heuristics do not produce a single next best move, process the *variables and values in lexicographical order*. The assignment of a value to a variable is depicted with "(a)".

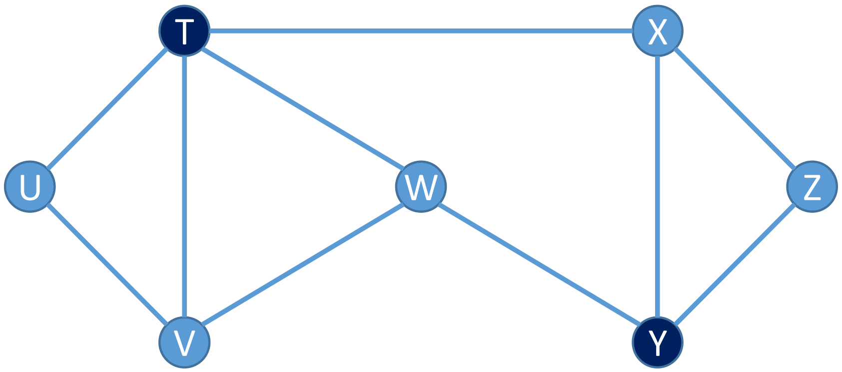
Exercise 1.5 – Solution

	T	U	V	W	X	Y	Z
Init	B,G,R	B,G,R	B,G,R	B,G,R	B,G,R	B,G,R	B,G,R
1	B (a)	G,R	G,R	G,R	G,R	B,G,R	B,G,R
2							
3							
4							
5							
6							
7							
8							



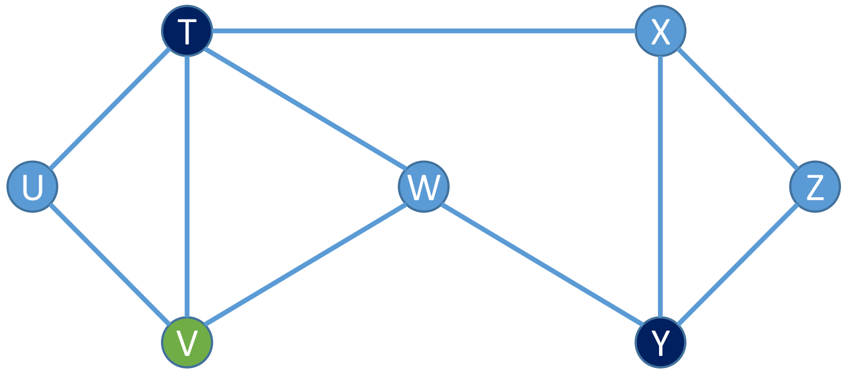
Exercise 1.5 – Solution

	T	U	V	W	X	Y	Z
Init	B,G,R	B,G,R	B,G,R	B,G,R	B,G,R	B,G,R	B,G,R
1	B (a)	G,R	G,R	G,R	G,R	B,G,R	B,G,R
2		G,R	G,R	G,R	G,R	B (a)	G,R
3							
4							
5							
6							
7							
8							



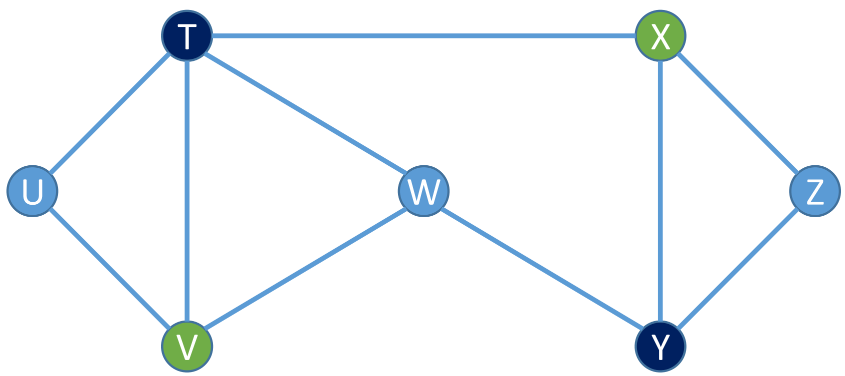
Exercise 1.5 – Solution

	T	U	V	W	X	Y	Z
Init	B,G,R	B,G,R	B,G,R	B,G,R	B,G,R	B,G,R	B,G,R
1	B (a)	G,R	G,R	G,R	G,R	B,G,R	B,G,R
2		G,R	G,R	G,R	G,R	B (a)	G,R
3		R	G (a)	R	G,R		G,R
4							
5							
6							
7							
8							



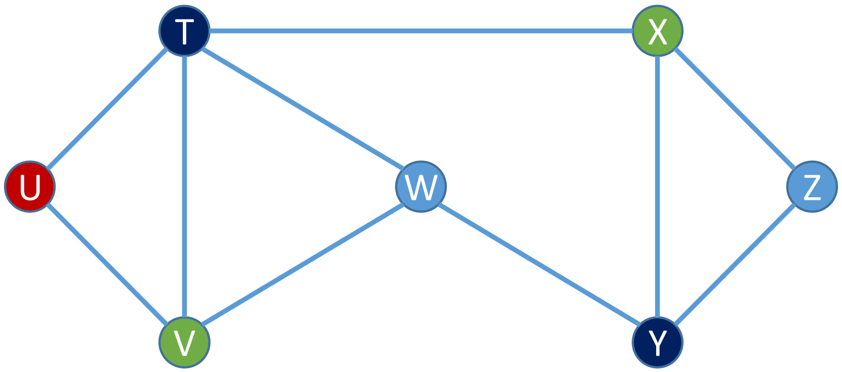
Exercise 1.5 – Solution

	T	U	V	W	X	Y	Z
Init	B,G,R	B,G,R	B,G,R	B,G,R	B,G,R	B,G,R	B,G,R
1	B (a)	G,R	G,R	G,R	G,R	B,G,R	B,G,R
2		G,R	G,R	G,R	G,R	B (a)	G,R
3		R	G (a)	R	G,R		G,R
4		R		R	G (a)		R
5							
6							
7							
8							



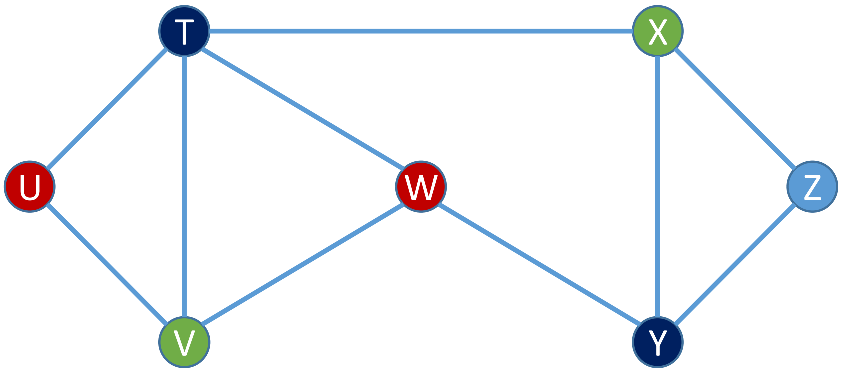
Exercise 1.5 – Solution

	T	U	V	W	X	Y	Z
Init	B,G,R	B,G,R	B,G,R	B,G,R	B,G,R	B,G,R	B,G,R
1	B (a)	G,R	G,R	G,R	G,R	B,G,R	B,G,R
2		G,R	G,R	G,R	G,R	B (a)	G,R
3		R	G (a)	R	G,R		G,R
4		R		R	G (a)		R
5		R (a)		R			R
6							
7							
8							



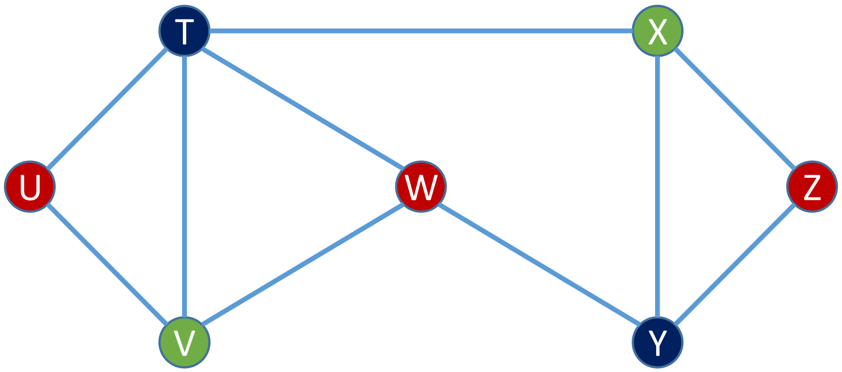
Exercise 1.5 – Solution

	T	U	V	W	X	Y	Z
Init	B,G,R	B,G,R	B,G,R	B,G,R	B,G,R	B,G,R	B,G,R
1	B (a)	G,R	G,R	G,R	G,R	B,G,R	B,G,R
2		G,R	G,R	G,R	G,R	B (a)	G,R
3		R	G (a)	R	G,R		G,R
4		R		R	G (a)		R
5		R (a)		R			R
6				R (a)			R
7							
8							



Exercise 1.5 – Solution

	T	U	V	W	X	Y	Z
Init	B,G,R	B,G,R	B,G,R	B,G,R	B,G,R	B,G,R	B,G,R
1	B (a)	G,R	G,R	G,R	G,R	B,G,R	B,G,R
2		G,R	G,R	G,R	G,R	B (a)	G,R
3		R	G (a)	R	G,R		G,R
4		R		R	G (a)		R
5		R (a)		R			R
6				R (a)			R
7							R (a)
8							



Exercise 1.5

Implement the class ColoringCSP and the additional methods in the provided .ipynb to solve coloring problems similar to the one presented in lecture 18. You should select variables and colors in lexicographical order.