Below you may find solutions to the sample exam. Note that the solutions are provided are one out of possibly many ways of answering the questions.

Algorithms for NP-Hard Problems
Sample Exam for Search and Inference

Emir Demirović

April 13, 2021

1 Exhaustive Search

We represent the solution as a vector of nodes $(n_1, n_2, ..., n_k)$ with the meaning that the cycle starts at node n_1, then moves to n_2, then to n_3, and so on. Therefore n_i is an integer variable where each integer represents a node. Assignments are only possible between two consecutive nodes if there is an edge.

- Start with $n_1 = D$.
- Decide $n_2 = C$.
- Decide $n_3 = M$. At this point nodes P, B, and L are pruned from n_4 because of the at most 30 constraint, backtrack.
- Decide $n_3 = L$. The node M is pruned from n_4 by the at most 30 constraint, and D cannot be selected, backtrack.
- Decide $n_3 = B$. Node M is pruned from n_4, there only P is left, and is assigned by propagation $n_4 = P$. However M is also pruned from n_5 by the at most 30 constraint, backtrack to n_2.
- Decide $n_2 = B$.
- Decide $n_2 = P$.

2 Modelling with Symmetry Breaking

- We introduce *binary* variables $X_{(i,j)}$ which denote that machine i is processing task j, and add constraints that ensure the capacity limit is not exceeded: $\forall i \in [1,2,3,4] : \sum_j R(t_j) \cdot X_{(i,j)} \leqslant C(m_i)$.
- There is a symmetry that arises because machines m_1 and m_4 have the same capacity, i.e., exchanging the assignments of tasks between the two machines leads to a symmetry solution, assuming at least one task was assigned to either machines m_1 or m_4. We can add the constraint $\sum_j R(t_j) \cdot X_{(1,j)} \leqslant \sum_j R(t_j) \cdot X_{(4,j)}$ to break this symmetry, imposing that machine m_1 will always have equal or less load than machine m_4.
- The search space size without the symmetry breaking constraint is 2^{4n}, where n is the number of tasks, since we have $4n$ binary variables. With symmetry breaking, we are removing roughly $(2^{4n} - 2^{2n})/2$ solutions, since only solutions where at least one task is allocated to m_1 or m_2 are symmetric, resulting in approximately a one-half search space reduction.
  - These statements could be made more precise, but this would not be necessary for the exam.

3 Propagation

- We represent the domain of the integer variables as an array of integer values.
- Arc consistency algorithm: for each value k in the domain of x, we test whether there is a corresponding value in the domain of y, namely the value $y = \sqrt{(k+5)/2}$. If there is no such value, then we may remove k from the domain of x. A similar procedure is done for each value in the domain of y, testing if a corresponding value exists in x.
- The domain of the integers explicitly represents all possible values. This is advantageous since we may filter out values efficiently using the above arc consistency algorithm and maintain an explicit list of possible values that may be assigned. The downside is that it may be too costly if the domain of the variables is large.

4 Look-Ahead

- We consider that each node is an integer variable, where each integer in the domain represents one colour.
- For each node n and each value v it may be assigned, we perform an assignment, and propagate. If there is at least one node with an empty domain, we may then remove value v from the domain of node n. Regardless of the outcome, we backtrack, effectively undoing the assignment of value v to node n.
- This allows us to detect forbidden values earlier in the search. Note that these propagations could not be detected by the standard graph colouring constraint since it only takes into account pairwise node and their assignments, and therefore may miss out on propagations that require reasoning over multiple nodes at once. The downside is that this type of look-ahead may be computationally expensive to perform at each iteration.

5 Relaxation

A possible relaxation is to temporarily ignore the Sudoku constraints, assign the largest numbers in the domains of the variables on the diagonals, and compute the sum of the values of the resulting diagonal. If the resulting value is smaller than the value in a reference solution (e.g., the best solution found so far), then the algorithm can immediately backtrack, since there is no hope of finding a better solution. The advantage of this relaxation is that it can prompt backtracks, saving computational time, and is relatively inexpensive to compute. The drawback is that it that the resulting values may be too optimistic, since all Sudoku constraints are ignored, and thus might not propagate often.