COMP 560 A1 KENKEN SOLVER

Zheng Qi, Hongyu Su, Julia Tian

Solution 1: Backtracking method (baseline)

we first read the input file and created a dictionary that mapped the letter with the operator, ie {“A”,”+”}. Then we also made a dictionary that mapped letter with corresponding coordinator, like{“A,(3,4)}.
Next we made a matrix so that we can assign corresponding value to it later. In addition, we keep the matrix which contains letter patterns, so that we can use coordinators to locate the specific letter constraint.
Now we created a node class that has fields of data: possible values for this node, next-node, previous-node, and corresponding coordinators.
For solution 1: we started with uppermost left node ie (0,0) and start with all possible values ie (1 to n). We looped through columns, ie (0,0)->(1,0)->...(0,1),(1,1)..->(n,n). So for all possible values for that coordinator(node), we first checked the row and column constraints, ie we removed repeated values in the same column and the same row. Then we checked the operator constraints. We based on coordinators to decide its corresponding operator and letter. In this way, we checked that whether the possible values satisfy operator constraints. We removed values that not satisfied and stored those values as a list in the data field of the node. We next checked whether the length of the data list is 0 or not, if it’s 0, we traced back to previous node until that length of that node is not 0 and assigned current Node.next=None. That means, we eliminated the link between the nodes. We popped out the first element in the current node data list (possible values) and assign the next element to the current coordinator. Then we find next node based on the column order and assign all corresponding next and previous field.

Solution 2: Best backtracking method with most constrained variable

we used the most constrained variable to optimize and we think it is the best backtracking method. For the normal backtracking method, we get around 2000 iterations while we use most constraints variable we get 900 iterations. We implemented this method a little bit similar with the first one. We added in some code. We check the nodes that are not filled. And for every empty nodes(coordinators) we calculated its possible values and chose the node that had the least number of possible values to be the next node. We then linked this next node with the current node. We ended the iterations when the length of unfilled nodes is 0. We implemented a dummy node to be the previous node of the first node. So if we tracked back to the dummy node, ie there is no possible value for even the first node, We threw exception that we can’t find the correct solution.

Solution 3: Local Search

By definition, a local search strategy retains no memory of previous states. Given a neighboring state that differs in the value of 1 cell, this solution utilizes a greedy approach to determine whether or not to transition to the next state or stay in the current one.

We randomly assign values within the domain to all cells, which we consider as the initial state. Then we randomly pick a cell, change the value and evaluate how it compares to the initial state.

The heuristic, or fitness function, used to guide the search is the number of violations given a fully populated KenKen puzzle. We use the simulated annealing optimization method to make decisions. If the next state contains less violations than our current state then we definitely consider the next state. If not, we take it with probability P(eenergy/temp)where temperature decreases over time and energy is just a measure of the difference between the next and current state.

Our ultimate goal is to reach the global minimum in which 0 constraints are violated. By the same logic, a local minimum is defined by the case where most constraints are fulfilled but there are still some violated. (Furthermore, there are no more downhill moves possible given this configuration.)

Since this solution could possibly get stuck in a local minimum where no further moves can be made, we allowed for up to N (size of board) restarts. Restarts are triggered randomly by a conditional that varies based on the size of the board.

Pseudocode:

```
For restarts 1 → N:
    If violations != 0:
        Initialize state
        Count initial violations
        For t → 1:
            Pick a cell and change its value
            Count updated violations
                If updated violations == 0 → done
                Else
                    If updated violations < initial violations
                        Transition to neighboring state
                    Else
                        Only transition to neighboring state with
                        P(eenergy/temp)
            Trigger random restarts
```

Statements of Individual Contribution

Zheng Qi: Pair-programmed the basic backtracking method and the best backtracking method with the most constrained variable with Hongyu Su
Hongyu Su: Pair-programmed the basic backtracking method and the best backtracking method with the most constrained variable with Zheng Qi
Julia Tian: Implemented the local search solution using greedy descent and a combination of simulated annealing and random restarts for optimization.