KAKURO – PUZZLE SOLVER GAME,

ALGORITHMIC APPROACHES

AND

ITS POSSIBLE APPLICATION TO REAL LIFE SCENARIO

ON FOOD TASTING

29.11.2014

STUDENT: MERVE KILINÇER – 150109028

SUPERVISOR: DR. MUSTAFA SAKALLI

1. Problem Statement

We repronounce an implication of Shakespeare as “Life is a tale and/or game and/or puzzle”. By saying this, in part we emphasize the importance of a daily challenge to survive, that is, the algorithms of determining the nutritional recipes of food and the taste induced in our sensory system cannot be different than the algorithms of games we pursue in everyday life. There must be a similarity between, due to the similarity of sensory and cognitive pathways. Therefore we determine to undertake a challenge of networking by providing a correspondence between the cognitive pathways of solving games and the pathways of for example food tasting as a real life scenario.
For this, initially, we adopt a reward based game, Kakuro-Puzzle Solver Game which is a puzzle solved by one player whose challenge level can be chosen by the user such that the complexity of the game can be extended from a linear complexity level (simple versions of the game) to NP complexity levels simply by increasing dimensions and the variety and redundancy of the different solution paths.

2. Problem Description

Games are an integral part of the human’s social life since the ancient times. Among all types of games, logic puzzles and board and games, historically have been considered very popular and were played not only to pass the time, but also to signify the wealth and status of the players. From scientific perspective, the logical puzzles have been analysed and studied from many mathematicians all over the world. Recently, with the technological development they have also become very attractive in the fields of computer science and artificial intelligence, since they have contributed to evaluation of performances of various metaheuristic algorithms and development of new ones.
Solving this kind of puzzles, besides the logic, often includes various constraints implied by the game grid, to which the solution should be fitted. It has been proven that many grid-based puzzles are non deterministic - complete problems i.e. they can accommodate cases requiring exponential times to be solved. On the other hand metaheuristic algorithms are considered an important alternative to bring up an approximate and satisfactory solutions to the problems. For example receptive and cognitive pathways of taste evaluation has similar complexities.
In this work we will be presenting a food or wine tasting analysis model by employing “Harmony Search Metahauristic Algorithm based” solution integrated with a popular grid-based Japanese game know as Kakuro. The proposed solution will be experimentally evaluated on various grids of presentations in different size and different applicable patterns.
To narrow down the problem, we will take the wine tasting.
The game we have chosen might look like a simple one but this is at least an initial stage of addressing algorithmic approaches to a complex cognitive process of taste, odor, aroma evaluation. The game Kakuro was utilized in other scenarios. We expect our approach to be just educational, but also it can be used as an application in real life scenario of conflict resolving or quality maintaining.

3. Preliminary Plan of the Project

- **Step 1:** Creating a Kakuro-Puzzle **Solver** Game on Android OS at the moment. (Almost completed.)
- **Step 2:** Using Metaheuristics such as "Harmony Search Metaheuristic Algorithm", Taboo Search and ant algorithms in correspondence to seek an approximate solution of such in complex cases of the game (Not complete).
- **Step 3:** Initially we aim to complete the puzzle/game for increasing number of dimensions such that its difficulty can be (linearly) adjusted by the user because we aim a reward (not a regret) based game. There were some challenges to be listed down.
- **Step 4:** Final project aim is that, as mentioned above to extend the scenario of the puzzle to a real life application, which may be educational or may be utilized for completely promoting a product or choosing a most favored one successfully as a result, by using controlling parameters to be determined for analysing food or wine tasting.
- **A block diagram of Kakuro Puzzle Creator and Solver is given in Figure.1**
- **A block diagram of Food Tasting and Smelling is given in Figure.2**

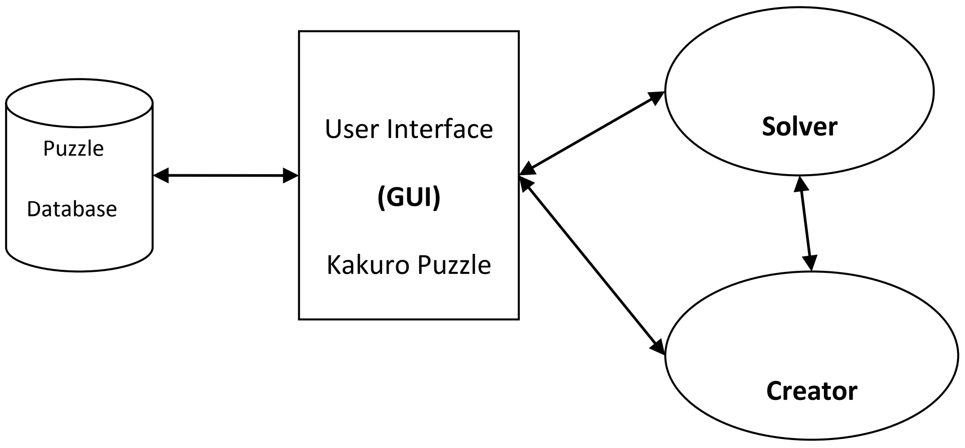


Figure.1: Block diagram of Kakuro Puzzle Creator and Solver

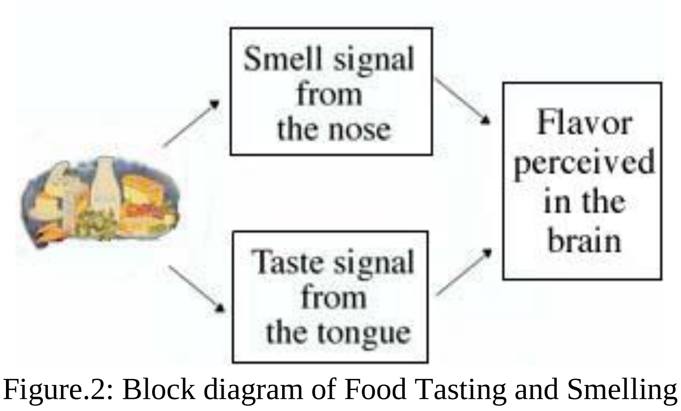

Smell signal from the nose
Taste signal from the tongue
Flavor perceived in the brain

Figure.2: Block diagram of Food Tasting and Smelling

4. Related Work

Related Work - Kakuro:

There have been several previous attempts for solving the Kakuro puzzle in the past few years. The NP-completeness of the puzzle has been proved using various techniques (Seta, 2002), and some of the latest proofs include using SAT solvers (Ruepp and Markus, 2010). Some researches (Davis et al., 2009) provided methods using heuristic approaches to improve a backtracking solving technique and introduced a pruning approach to decrease the time of obtaining a solution. They have also emphasized the utilization of these techniques in coding theory and applying them in real-world applications, which proved the importance of developing such algorithms.
A Kakuro puzzle can be easily observed as a constrained optimization problem (Simonis, 2008). It has been proven that by using generalized arc consistent (GAC) version of all-different sums constraint reduces the necessity of searching, compared to MIP (Achterberg et al., 2006) and SAT techniques (Eén and Sörensson, 2004). For the first time it has been shown that a Nested Monte-Carlo search of level 2 can give approximate satisfying results for the purposes of solving such puzzles (Cazenave, 2010).
Harmony Search (HS) is an optimization technique which is analogous to the process of improvisation of Jazz musicians (Yang, 2009) and has been widely used in the last decade for optimization purposes. HS has already been used for solving Sudoku puzzles (Geem, 2007) and has proved to provide good solutions in 285 iterations in less than 9 seconds. This is a motivation to apply HS to other challenging NP-complete puzzles, as we chose it as Kakuro, as described below.

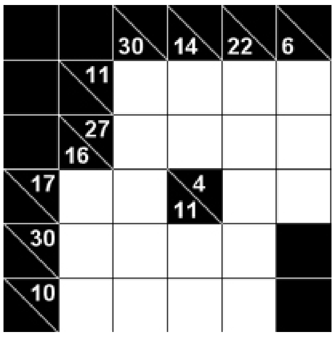
Related Work - Food Tasting Analysing:

As presented in various articles of Smith's and Allhoff's books, indeed, Jamie Goode, as a distinguished wine critic, trained biochemist and contributor to both volumes, has a surprisingly semiotic argument. In his article "Wine and the Brain" (2007), Goode states:
"I'm going to argue (perhaps controversially) that the way rating or scoring wine is currently practised is based on a false premise: that when a critic rates a wine, they are assessing the wine, and that any score thus produced is a property of the wine. This is incorrect. We need to make the subtle yet important paradigm shift of seeing a critic's assessment as a rating of that critic's perception of the wine. To put it another way, the critic is actually describing a conscious representation of their interaction with the wine, and therefore the score or rating is a property of that interaction and not the wine itself.".
Yet, there has not been a related work analysing food or wine tasting as an optimization problem in computer science. But we determine to achieve a model of analysing the sense of taste by using algorithmic approaches of Kakuro-Puzzle Game Solver.

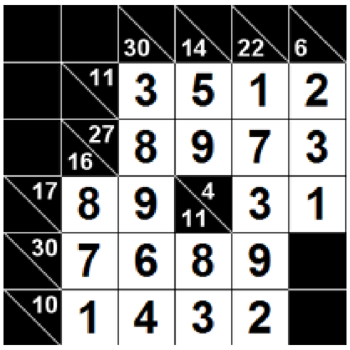
5. **Kakuro-Puzzle / Game**

Preliminaries and Concept:

Kakuro is a type of grid-based logic puzzle that is usually known as a mathematical transliteration of the crossword. It is composed of a simple grid which dimensions can vary from 3x3 matrix through to a grid with large dimensions NxM (N,M >= 3). The grid is composed of “blank” and “occupied” cells, white and black correspondently. By rule the top row and leftmost column are occupied and are not used for the game while the rest or the grid contains entry series of white cells divided by some black cells. Some of the black cells contain a diagonal slash from upper-left to lower-right corner and a number in one or both halves, denominated as clues (as shown below).



The main aim of the game is to fill in all the white cells with numbers from 1 to 9. All cells in the same row and all cells in the same column have to contain different number. The sum of the numbers of a row or a column has to match a predefined clue, associated with it.
The puzzle is solved when all the white cells are filled in according to the game rules and respecting the given constraints. The solution of the above Kakuro puzzle is presented in below image. The solution of the game mainly depends on its structure, and usually more blank cells the grid contains, the harder it is to be solved.



Various solving techniques of Kakuro puzzles exists and some of them include: Lone square, Cross reference, Combo reference, Filled areas, Eliminating duplicates. There are two kinds of mathematical symmetry readily identifiable in kakuro puzzles: minimum and maximum constraints are duals, as are missing and required values.

Is Kakuro NP?

An integer programming problem is an optimization problem of the form

$$\textbf{extremize}\ f(x_1,x_2,...,x_D) \quad \text{subject to } L_i(x_1,x_2,...,x_D)<0$$

where L_i and f are given functions, D, is the number of variables, which is the dimension of the null space of the linear algebra problem. The Kakuro problem can be mapped into this in the following way.

Since there are M white cells in the problem, then there are M Type 1 constraints. These can be written in the form of 2M linear constraints in the standard form $L_i \le 0$. The constraints become $C.x \le 0$ where C is a $2M \times D$ matrix, which is possibly reducible, since some of the constraints could be identical. The cost function can be created from the Type 2 inequalities $lk(x_j) \neq rk(x_j)$, by squaring the difference $lk(x_j)-rk(x_j)$, and summing the squares over all constraints. Since no clue in Kakuro can involve partitions into more than 9 pieces therefore the number of Type 2 constraints must be between M and 4M. The number can reduce if some of the constraints are identical or are obviously satisfied by integers.

We have $f(x_j) = \sum_{j=1:K} [lk(x_j)-rk(x_j)]^2$, where K is a number linear in M.

The Kakuro problem is the non-linear integer programming problem

$$\textbf{maximize}\ f(x_j) = \sum_{j=1:K} [lk(x_j)-rk(x_j)]^2 \quad \text{subject to } C.x \le 0.$$

Since the constraints are linear, they make up the faces of a polyhedron in the D dimensional space. The integer programming problem can be solved, in principle, by visiting every integer point inside this polyhedron and computing the cost function there. The general integer programming problem is known to be NP-hard. In other words, one can check a claimed solution for D variables, in a number of steps of arithmetic which is less than some fixed power of D. However, generating such a solution takes an amount of arithmetic larger than any fixed power of D.

Is Kakuro in NP? The spreadsheet part of the algorithm takes time of order M and gives D free variables. The integer programming problem in these D variables can be solved by enumeration in at most 9D steps, since each variable can take at most 9 values; this certainly grows faster than any fixed power of D. On the other hand, if one were given a claimed solution, one could check it by filling this back into the spreadsheet in a time of order M. Since D is not exponentially smaller than M, an exponential in D cannot be polynomial in M. This shows that if enumeration were the only way of solving Kakuro puzzles, then generating a solution of Kakuro would take non-polynomially large time. In this case Kakuro would be in NP.

Harmony Search Metaheuristic Algorithm:

In the Harmony Search (HS) algorithm, an optimal solution is constructed by using all available combinations present in memory, i.e. the Harmony Memory (HM), and then uses randomization techniques to select candidate values to obtain new solutions towards the optimal value of a fitness function. HS constructs the best possible solution in several steps:

1) <u>Harmony Memory (HM) Initialization:</u>
   The matrix of all possible solutions has to be initialized to randomly generated values, all specified in a predefined range of numbers.

2) Candidate Solution Improvisation:
   This step of the process is dependent of several parameters. The first is Harmony Memory Consideration Rate (HMCR), which indicates the probability the next constructed solution to be selected from HM, whereas (1-HMCR) would indicate the probability this new solution to be generated randomly from the range of possible values. The next crucial parameter, if one needs a certain variability of the chosen solutions and ability to influence the memory values, is the Pitch Adjustment Rate (PAR). Thus, PAR indicates the probability of a candidate solution being subject to change. Furthermore, the maximum permitted variability in pitch adjustment is given by the bandwidth (BW) parameter.
3) Comparison:
   The new constructed solution is then compared to all the vectors present in the HM. If this new vector provides better fitness function evaluation then the worst present solution in HM, these two solutions are exchanged, so the new solution is temporarily inserted in memory, and the worst is discarded. If the value of the fitness function does not satisfy the previously stated condition, the new candidate solution is discarded. If the termination criteria is not met, such as number of iterations or a concrete value of the fitness function, steps 2) and 3) are repeated. Otherwise, the algorithm terminates.

Creating Kakuro-Puzzle Solver Game On Android OS:

We haven't implemented the Kakuro-solver yet. We will implement the solver with additional options, increasing complexity and symbolic presentations. We are considering to extend it to the specific food industry by adapting different symbols. Kakuro-Puzzle Solver Game on Android OS will be implemented by using Java Programming Language with Eclipse IDE as mentioned Harmony Search Metaheuristic Algorithm while developing this game.
We thought about how we would solve a Kakuro puzzle, then develop the procedures, and get the program to move through the puzzle (backtracking by recursively calling our procedures).
One thing that we would consider for Kakuro puzzles is to get the program to store possible number combinations from rows and columns and cross-check each row and column to see if any numbers are required to be somewhere, then get the program to try to solve from there using normal logic. All these will require some careful programming effort and logic to be dedicated.

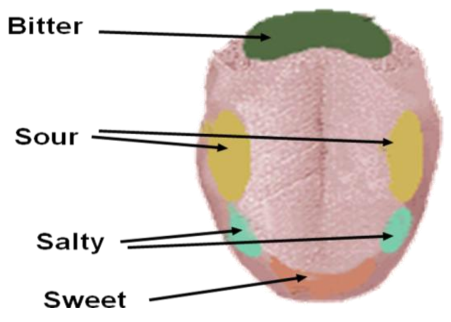
About Food Tasting Analysis:

Taste or flavor characteristics of foods were not very important in the 'selection factors' used by plant breeders until recently. But now, recent studies suggest four fundamental parameters contributing overall taste sensing:

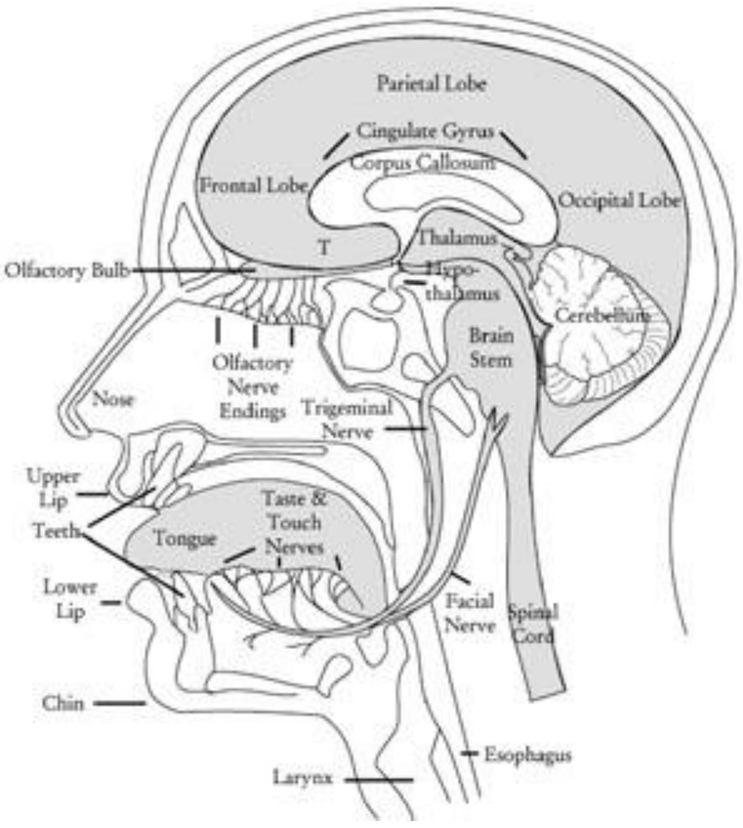
- Sweet
- Sour
- Salty
- Bitter

These are parameters sensed in specific locations of the tongue.

Taste sensation areas on the tongue



From Tongue To Brain:



Our initial approach will involve orthogonalising the data to achieve nonovelapping paths of perception, sensory paths to make a sense of it, as a combinatorial combination of those, Taste determined by the tongue and Aroma and/or Smell.
While we can only determine some taste factors, our assumption is that we can differentiate just 4 taste factors, although in reality there is a large number of aromas or flavors in foods (<=10,000) that can be differentiated. So, we assumed that each flavor is a path of Kakuro Puzzle Solver

which would require a large dimension to be adopted, but for the sake of simplicity we take just 4 of which.

6. Success Factors and Benefits

As we mentioned above, with the four taste factors perceived over 10,000 aromas or flavors, we implement a 4x4 Kakuro Solver. It is true that almost 10,000 paths will be needed in the application of Kakuro Solver that remains to be tested in further stages.
In Kakuro, the placement of many values will lead to violations of the puzzle constraints – a duplicate value in a run, or an exceeded or under-target run-total on completion of the final cell of a run. Such violations can be used to prune evidently fruitless branches.
This puzzle solver will be used to determine the tastes.

7. Management Plan

TASK	DESCRIPTION	STATUS
Researching the game Kakuro Puzzle.	Answer these questions: ”What is Kakuro, game or puzzle?” “Mathematical frame of solving Kakuro Puzzle?” “How to create a Kakuro Puzzle **Solver**?”	Partially completed.
Improving ourselves on Java development in Android OS.	Also before take Graduation Project Course, we were getting trained about that. Due to the course on “Mobile Device Developing” and our trainings, this will be a practice for me.	In progress.
Implementing Kakuro Puzzle Solver	By using Java programming language and Android SDK	In progress
Researching about food tasting.	Answer these questions: “Which parameters to analyse tastes?” “How does our brain determine the tastes?” “Algorithms.. Which tastes are determined by us?	Not completed yet
Anlysing related works.	If any in literature, not discovered yet.	Not completed yet

Programs developed are presented in Appendix A.

8. Methodology and Technical Approach

We thought about how we would solve a kakuro puzzle, then develop the procedures, and get the program to move through the puzzle (probably by recursively calling our procedures).
One thing that we would consider for kakuro puzzles is to get the program to store possible number combinations from rows and columns and cross-check each row and column to see if any numbers are required to be somewhere, then get the program to try to solve from there using normal logic.
It is taking some effort requiring careful programming to lay down a precise logic underneath. To solve this problem, we are in progress of mathematical modeling of Kakuro and such games and elaborating one to one mapping of such.

9. References

Concepts Puzzles (2006). Black Belt Kakuro. Sterling Publishing Co., Inc, Toronto, Ontario, Canada.

Davies, R. P. (2009). An Investigation into the Solution to, and Evaluation of, Kakuro Puzzles, MSc thesis, University of Glamorgan.

Simonis, H., (2008). Kakuro as a Constraint Problem. In: Proceedings of Modref Conference, 2008.

A. Chisholm. How to solve Kakuro puzzles. 2005. http://www.indigopuzzles.com/ipuz/.

Friedman, E.: Corral puzzles are NP-complete. Technical report, Stetson University, DeLand, FL 32723 (2002)

Stamp, M. (2012). A Revealing Introduction to Hidden Markov Models [PDF document]. Retrieved from http://www.cs.sjsu.edu/~stamp/RUA/HMM.pdf.

Ramage, D. (2007). Hidden Markov Models Fundamentals [PDF document]. Retrieved from http://cs229.stanford.edu/section/cs229-hmm.pdf.

Deen, Aaron D. "Food And Wine Pairing." Food and Wine Pairing. N.p., 2012. Web. 17 Mar. 2013. http://www.foodandwinepairing.org/wine_pairing_board.html

Hutchins, Max O. "Chemical Senses: Olfaction and Gustation." Neuroscience Online. The University of Texas Medical School at Houston, 6 Mar. 1997. Web. 04 Apr. 2013. http://neuroscience.uth.tmc.edu/s2/chapter09.html

Nase, Joseph. "The Four Most Common Defects and How to Detect Them." The Sommelier.

New York Magazine, 2013. Web. 13 Mar. 2013.
http://nymag.com/restaurants/articles/wine/essentials/badwine.htm.

"The Science of Taste." Kitchen Geekery. N.p., 2013. Web. 17 Mar. 2013.
http://www.kitchengeekery.com/articles/science/the-science-of-taste

10. Appendix

```
public class Kakuro extends AbstractProblem {
  // size of matrix
  int n = 7;
  // Segments:
  // sum, the segments
  // Note: this is 1-based (fixed below)
  int[][] problem =
  {
    new int[] {16,  1,1, 1,2},
    new int[] {24,  1,5, 1,6, 1,7},
    new int[] {17,  2,1, 2,2},
    new int[] {29,  2,4, 2,5, 2,6, 2,7},
    new int[] {35,  3,1, 3,2, 3,3, 3,4, 3,5},
    new int[] { 7,  4,2, 4,3},
    new int[] { 8,  4,5, 4,6},
    new int[] {16,  5,3, 5,4, 5,5, 5,6, 5,7},
    new int[] {21,  6,1, 6,2, 6,3, 6,4},
    new int[] { 5,  6,6, 6,7},
    new int[] { 6,  7,1, 7,2, 7,3},
    new int[] { 3,  7,6, 7,7},
    new int[] {23,  1,1, 2,1, 3,1},
    new int[] {30,  1,2, 2,2, 3,2, 4,2},
    new int[] {27,  1,5, 2,5, 3,5, 4,5, 5,5},
    new int[] {12,  1,6, 2,6},
    new int[] {16,  1,7, 2,7},
    new int[] {17,  2,4, 3,4},
    new int[] {15,  3,3, 4,3, 5,3, 6,3, 7,3},
    new int[] {12,  4,6, 5,6, 6,6, 7,6},
    new int[] { 7,  5,4, 6,4},
    new int[] { 7,  5,7, 6,7, 7,7},
    new int[] {11,  6,1, 7,1},
    new int[] {10,  6,2, 7,2}
  };
  int num_p = problem.length; // Number of segments
  // The blanks
  // Note: 1-based
  int[][] blanks = {
    {1,3}, {1,4},
    {2,3},
    {3,6}, {3,7},
    {4,1}, {4,4}, {4,7},
    {5,1}, {5,2},
    {6,5},
    {7,4}, {7,5}
  };
  int num_blanks = blanks.length;
  IntVar[][] x;
  @Override
  public void buildModel() {
    x = VariableFactory.enumeratedMatrix("x", n, n, 0, 9, solver);
    // fill the blanks with 0
    for(int i = 0; i < num_blanks; i++) {
      solver.post(IntConstraintFactory.arithm(x[blanks[i][0]-1][blanks[i][1]-1], "=", 0));
    }
    for(int i = 0; i < num_p; i++) {
      int[] segment = problem[i];
      // Remove the sum from the segment
      int[] s2 = new int[segment.length-1];
      for(int j = 1; j < segment.length; j++) {
        s2[j-1] = segment[j];
      }
      // all numbers in this segment must be distinct
      int len = segment.length / 2;
      IntVar[] t = VariableFactory.enumeratedArray("t", len, 1, 9, solver);
      for(int j = 0; j < len; j++) {
        t[j] = x[s2[j * 2] - 1][s2[j * 2 + 1] - 1];
        // ensure that the values are positive
        solver.post(IntConstraintFactory.arithm(t[j],">=", 1));
```

```java
      }
      solver.post(IntConstraintFactory.alldifferent(t, "BC"));
      // sum the segment
      solver.post(IntConstraintFactory.sum(t, VariableFactory.fixed(segment[0], solver)));
    }
  }
  @Override
  public void createSolver() {
    solver = new Solver("Kakuro");
  }
  @Override
  public void configureSearch() {
    solver.set(IntStrategyFactory.firstFail_InDomainMin(ArrayUtils.flatten(x)));
  }
  @Override
  public void solve() {
    solver.findSolution();
  }
  @Override
  public void prettyOut() {
    if (solver.isFeasible() == ESat.TRUE) {
      int num_solutions = 0;
      do {
        for(int i = 0; i < n; i++) {
          for(int j = 0; j < n; j++) {
            int v = x[i][j].getValue();
            if (v > 0) {
              System.out.print(v + " ");
            } else {
              System.out.print("_ ");
            }
          }
          System.out.println();
        }
        System.out.println();
        num_solutions++;
      } while (solver.nextSolution() == Boolean.TRUE);
      System.out.println("It was " + num_solutions + " solutions.");
    }  else {
      System.out.println("No solution.");
    }
  }
  public static void main(String args[]) {
    new Kakuro().execute(args);
  }
}
```