**EECS 543 – Project 3 Analysis and Evaluation**

Jason Varbedian (jpvarbed), Dan Jonik (djonik)

**Task 1: Defining KenKen**

We defined our KenKen puzzle in terms of cell objects and constraint objects. Each cell stores a coordinate pair, a list of possible values for the cell, the constraint that that cell must satisfy, and finally a list of all of the cell’s neighbors. The constraint object has an operator, the expected outcome, and the list of cells that together must satisfy the constraint value. The puzzle itself is primarily a 2D array of cell objects.

Additionally, there is error checking to make sure that all coordinates listed are legal and that each cell is listed in exactly one constraint region.

Below is a printout of a simple puzzle data structure:

>> (SETF 2PUZ (CREATE-PUZZLE 2 '((3 + ((1 1) (1 2))) (1 + ((2 1))) (2 + ((2 2))))))

#S(PUZZLE :CELLS #2A((

#S(CELL :X 1

:Y 1

:DOMAIN (1 2)

:CONSTRAINT #S(CONSTRAINT :OUTCOME 3 :OPERATION + :REGION-CELLS #)

:NEIGHBORS ...)

#S(CELL :X 1

:Y 2

:DOMAIN (1 2)

:CONSTRAINT #S(CONSTRAINT :OUTCOME 3 :OPERATION + :REGION-CELLS #)

:NEIGHBORS ...))

(#S(CELL :X 2

:Y 1

:DOMAIN (1 2)

:CONSTRAINT #S(CONSTRAINT :OUTCOME 1 :OPERATION + :REGION-CELLS #)

:NEIGHBORS ...)

#S(CELL :X 2

:Y 2

:DOMAIN (1 2)

:CONSTRAINT #S(CONSTRAINT :OUTCOME 2 :OPERATION + :REGION-CELLS #)

:NEIGHBORS ...)))

:SIZE 2

:CONSTRAINTS (#S(CONSTRAINT :OUTCOME 3 :OPERATION + :REGION-CELLS ((1 1) (1 2)))

#S(CONSTRAINT :OUTCOME 1 :OPERATION + :REGION-CELLS ((2 1)))

#S(CONSTRAINT :OUTCOME 2 :OPERATION + :REGION-CELLS ((2 2)))))

**Task 2: Solving KenKen (print-solutions)**

*Some of the formatting has been changed to fit tests on individual pages.*

Test 1: a simple 2x2 puzzle that can be solved by simply propagating constraints (no searching)

>> (setf 2puz (setf 2puz (create-puzzle 2 '((3 + ((1 1) (1 2))) (1 + ((2 1))) (2 + ((2 2))))))

>> (print-solutions 2puz)

Arithmetic Constraints:

12

----

2|AA|

3|BC|

----

Constraint A: 3+ ((1 1) (1 2))

Constraint B: 1+ ((2 1))

Constraint C: 2+ ((2 2))

After constraint propagation the puzzle is:

Puzzle:

12

----

2|21|

3|12|

----

Domain Sizes:

12

----

2|11|

3|11|

----

Searching...

There is one solution:

Puzzle:

12

----

2|21|

3|12|

----

Test 2: a 3x3 KenKen puzzle that has two correct solutions

>>(setf 3puz (create-puzzle 3 '((6 + ((1 1) (2 1) (3 1))) (6 \* ((1 2) (1 3) (2 3))) (3 + ((2 2))) (1 - ((3 2) (3 3))))))

>> (print-solutions 3puz)

Arithmetic Constraints:

123

-----

2|ABB|

3|ACB|

4|ADD|

-----

Constraint A: 6+ ((1 1) (2 1) (3 1)) Constraint B: 6\* ((1 2) (1 3) (2 3)) Constraint C: 3+ ((2 2))

Constraint D: 1- ((3 2) (3 3))

After constraint propagation the puzzle is:

Puzzle:

123

-----

2|...|

3|.3.|

4|...|

-----

Domain Sizes:

123

-----

2|323|

3|212|

4|323|

-----

Searching...

There are two solutions:

Puzzle:

123

-----

2|123|

3|231|

4|312|

-----

Puzzle:

123

-----

2|213|

3|132|

4|321|

-----

Test 3: a 5x5 KenKen that has no subregions containing a single cell. This causes the initial constraint propagation to be completely ineffective, so there is a considerable amount of searching to find the solution.

>> (setf 5puz (create-puzzle 5 '((10 + ((1 1) (1 2) (1 3) (1 4))) (10 + ((1 5) (2 5) (3 5)))

(10 + ((2 1) (2 2) (3 1))) (10 \* ((2 3) (2 4) (3 4))) (10 + ((3 2) (3 3) (4 2)))

(10 + ((4 1) (5 1) (5 2))) (10 + ((4 3) (5 3) (5 4) (5 5))) (10 \* ((4 4) (4 5))))))

>> (print-solutions 5puz)

Arithmetic Constraints:

12345

-------

2|AAAAB|

3|CCDDB|

4|CEEDB|

5|FEGHH|

6|FFGGG|

-------

Constraint A: 10+ ((1 1) (1 2) (1 3) (1 4))

Constraint B: 10+ ((1 5) (2 5) (3 5))

Constraint C: 10+ ((2 1) (2 2) (3 1))

Constraint D: 10\* ((2 3) (2 4) (3 4))

Constraint E: 10+ ((3 2) (3 3) (4 2))

Constraint F: 10+ ((4 1) (5 1) (5 2))

Constraint G: 10+ ((4 3) (5 3) (5 4) (5 5))

Constraint H: 10\* ((4 4) (4 5))

After constraint propogation the puzzle is:

Puzzle:

12345

-------

2|.....|

3|.....|

4|.....|

5|.....|

6|.....|

-------

Domain Sizes:

12345

-------

2|55555|

3|55555|

4|55555|

5|55555|

6|55555|

-------

Searching...

There is one solution:

Puzzle:

12345

-------

2|21435|

3|32514|

4|54321|

5|43152|

6|15243|

-------

Test 4: a stress test case featuring a 6x6 KenKen with 18 constraint regions

>> (SETF 6PUZ (CREATE-PUZZLE 6

'((80 \* ((1 1) (1 2) (2 1))) (3 + ((1 3))) (5 - ((1 4) (1 5)))

(2 / ((1 6) (2 6))) (11 + ((2 2) (2 3))) (1 - ((2 4) (2 5)))

(9 \* ((3 1) (4 1) (4 2))) (2 + ((3 2))) (3 - ((3 3) (3 4)))

(30 \* ((3 5) (3 6))) (11 + ((4 3) (4 4))) (2 / ((4 5) (4 6)))

(6 + ((5 1))) (10 \* ((6 1) (6 2))) (8 \* ((5 2) (5 3) (6 3)))

(13 + ((5 4) (6 4) (5 5))) (1 + ((6 5))) (8 + ((5 6) (6 6))))))

>>(print-solutions 6puz)

Arithmetic Constraints:

123456

--------

2|AABCCD|

3|AEEFFD|

4|GHIIJJ|

5|GGKKLL|

6|MOOPPR|

7|NNOPQR|

--------

Constraint A: 80\* ((1 1) (1 2) (2 1))

Constraint B: 3+ ((1 3))

Constraint C: 5- ((1 4) (1 5))

Constraint D: 2/ ((1 6) (2 6))

Constraint E: 11+ ((2 2) (2 3))

Constraint F: 1- ((2 4) (2 5))

Constraint G: 9\* ((3 1) (4 1) (4 2))

Constraint H: 2+ ((3 2))

Constraint I: 3- ((3 3) (3 4))

Constraint J: 30\* ((3 5) (3 6))

Constraint K: 11+ ((4 3) (4 4))

Constraint L: 2/ ((4 5) (4 6))

Constraint M: 6+ ((5 1))

Constraint N: 10\* ((6 1) (6 2))

Constraint O: 8\* ((5 2) (5 3) (6 3))

Constraint P: 13+ ((5 4) (6 4) (5 5))

Constraint Q: 1+ ((6 5))

Constraint R: 8+ ((5 6) (6 6))

After constraint propagation the puzzle is:

Puzzle:

123456

--------

2|..3...|

3|......|

4|.2....|

5|......|

6|6.....|

7|....1.|

--------

Domain Sizes:

123456

--------

2|441545|

3|555656|

4|414545|

5|555656|

6|144545|

7|444515|

--------

Searching...

There is one solution:

Puzzle:

123456

--------

2|543162|

3|465231|

4|321456|

5|136524|

6|612345|

7|254613|

--------

**Tasks 3 and 4: mrv-ambiguous, extended-consistency**

**Number of Guesses Made**

|  |  |  |  |
| --- | --- | --- | --- |
| **Puzzle** | **first-ambiguous** | **mrv-ambiguous** | **Extended-consistency (default search-heuristic)** |
| 2puz | 0 | 0 | 0 |
| 3puz | 5 | 2 | 2 |
| 4puz | 25 | 25 | 2 |
| 5puz | 1299 | 1431 | 78 |
| 6puz | 54 | 32 | 0 |

*mrv-ambiguous*

From the table above, we can observe that in the majority of cases, using *mrv-ambiguous* as the search-heuristic results in a fewer number of guesses while solving a KenKen puzzle. The exception to this is the 5puz test case; however, this is an unusual puzzle in that no constraints are able to be propagated before searching starts. Therefore, all cells initially have a domain of size five. The number of guesses between the two search heuristics is on the same order of magnitude; it appears to be a coincidence that the order in which *first-ambiguous* guessed values happened to find the correct solution faster than the order defined by *mrv-ambiguous*.

As would be expected, the memory and time performance of *print-solutions* using the two heuristics follows the same pattern as the number of guesses. Execution time was faster in cases that had fewer guesses; additionally, fewer guesses resulted in less memory being used. The fact that execution time decreased when using *mrv-ambiguous* shows that the time saved by making fewer guesses outweighs additional time spent running the *mrv* algorithm, instead of the simpler *first-ambiguous*.

*extended-consistency*

As seen by the example above, adding the *extended-consistency* feature greatly reduced the number of guesses made by *print-solutions*; however, it appeared to have a detrimental effect on runtime. Every time we called our function to remove impossible values from a cell’s domain, we enumerated through all possibilities of all values from the domains of the cells in a constraint region. As the puzzle size increases (resulting in the domain of possible values increasing), enumerating all of the constraint input possible became exponentially more expensive. The runtime of our two larger test cases shows this trend:

|  |  |  |  |
| --- | --- | --- | --- |
| **Puzzle** | **first-ambiguous** | **mrv-ambiguous** | **extended-consistency** |
| 5puz | 1.238000 sec | 0.941000 sec | 1.745000 sec |
| 6puz | 0.650000 sec | 0.547000 sec | 1.140000 sec |

Therefore, *extended-consistency* is worth using when either a) runtime is not a major factor, or b) we are able to successfully decrease the domains of all cells in the first several rounds of constraint propagation so that we do not have so many possible value pairs to evaluate for each constraint.