CS170 Project Writup

Eamonn Carson

December 2, 2017

1 Reduction

We reduce a Wizards Problem to a Boolean Satisfiability (SAT) Problem like so:

Let $R_{a,b}$ be a boolean variable. $R_{a,b}$ is true if and only if wizard a comes before wizard b in the optimal ordering. Note that in our case — since wizards have distinct ages — $R_{a,b} \iff \neg R_{b,a}$.

(Note that there may be multiple optimal orderings, but we ensure all $R_{a,b}$ refer to the same optimal ordering via the transitivity relationship SAT clauses later, so it is not relevant.)

Let $C =$ "a, b, c" be a clause of the wizards problem, then C is fulfilled if and only if we have $\neg(R_{a,c} \wedge R_{c,b}) \wedge \neg(R_{b,c} \wedge R_{c,a})$. However

$$\begin{aligned}\neg(R_{a,c} \wedge R_{c,b}) \wedge \neg(R_{b,c} \wedge R_{c,a}) &= (\neg R_{a,c} \vee \neg R_{c,b}) \wedge (\neg R_{b,c} \vee R_{c,a}) \\ &= (R_{c,a} \vee R_{c,b}) \wedge (\neg R_{b,c} \vee \neg R_{c,a})\end{aligned}$$

Let R be a set of boolean variables such that for all $a \neq b$ either $R_{a,b} \in R$ (exclusive or) or $R_{b,a} \in R$.

Then we may describe the Wizards Problem clauses in terms of a SAT problem with variables R and the SAT clauses

- $(R_{c,a} \vee R_{c,b})$
- $(\neg R_{b,c} \vee \neg R_{c,a})$

For each Wizards clause $C =$ "a, b, c".

However, we are not done. We also must encode the transitivity relationships between the $R_{a,c}$. If we enumerate the set of wizards W by $\mathrm{f} : W \mapsto \mathbb{N}$ then we only need to add the SAT clauses:

- $R_{a,b} \wedge R_{b,c} \implies R_{a,c}$ (equivalent to the SAT clause $\neg R_{a,b} \vee \neg R_{b,c} \vee R_{a,c}$)
- $\neg R_{a,b} \wedge \neg R_{b,c} \implies \neg R_{a,c}$ (equivalent to the SAT clause $R_{a,b} \vee R_{b,c} \vee \neg R_{a,c}$)

for all a, b, c, such that $f(a) < f(b) < f(c)$.

2 The Algorithm

My algorithm basically resolves to reducing the Wizards Problem to a SAT problem, using one of the highly developed SAT libraries to solve, and then translating back.

Specifically, given a Wizards Problem W_{prob}

- Convert W_{prob} to an equivalent SAT problem S_{prob} via the reduction above.
- Solve S_{prob} using the SAT4J Core Library (link).
- Using the resulting truth values on all inequalities, sort the list of wizards to generate a valid ordering.

3 Citations

I used the SAT4J Core Library (link) in my project because it boasted an easy to use interface. I used it to solve the SAT problem that I reduced the Wizards Problem to. I did this because the SAT problem is old and much research has been dedicated to it, so any reasonably mature SAT library would probably be faster than me cobbling up my own naive solution to the SAT or the Wizards Problem.

Installation: Download the SAT4J Core Library version 2.3.5 (Click download link under "SAT4J Core" at http://sat4j.org/products.php) and use it as a library in the submitted Java project.

4 Codebase

(I took the code formatting from
https://stackoverflow.com/questions/3175105/writing-code-in-latex-document3175141)

4.1 Overview

NotBetweenSolver is the main class. What it does is it reads the input file in order to create a NotBetween-Problem instance. It then translates the NotBetweenProblem into a SatisfiabilityProblem. Finally it uses SAT4J to solve the SatisfiabilityProblem (SAT problem), and then returns the resulting output.

It is a bit more complicated than that because it has to encode transitivity relations but thats the gist of it. Enjoy!

4.2 Class: NotBetweenSolver

```
import org.sat4j.specs.ContradictionException;
import org.sat4j.specs.TimeoutException;

import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;

/**
 * Created by eamonncarson on 11/10/17.
 */
public class NotBetweenSolver {

    public static void main(String[] argv) {
        if (argv.length < 1) {
            System.out.println("Error: 1 command line argument for the filename of Wizards file
                required.");
            System.exit(1);
        }
        String inputFilename = "input/" + argv[0] + ".in";
        NotBetweenProblem nbp = new NotBetweenProblem(inputFilename);
        SatisfiabilityProblem sat = null;
        try {
            sat = new SatisfiabilityProblem(nbp);
        } catch (TimeoutException e) {
            System.out.println("Error: SAT algorithm has timed out.");
            System.exit(1);
        } catch (ContradictionException e) {
            System.out.println("Error: I screwed up.");
            System.exit(1);
        }

        String[] wizardOrder = nbp.convertToNames(sat.getSolution());
        System.out.println("Found valid order:");
        String output = "";

        for (String wizard : wizardOrder) {
            output += wizard + " ";
        }

        System.out.println(output);

        String outputFileName = "output/" + argv[0] + ".out";
        try (BufferedWriter bw = new BufferedWriter(new FileWriter(outputFileName))) {
            bw.write(output);
        } catch (IOException e) {
```

```
            System.out.println("SORRY: error writing file.");
        }
    }

}
```

4.3 Class: NotBetweenProblem

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.regex.Pattern;

import static java.lang.System.exit;

/**
 * Created by eamonncarson on 11/10/17.
 */
public class NotBetweenProblem {

    int numWizards;
    String[] names; // the set of wizard names by index
    Map<String, Integer> nameId; // a map from wizard names to wizard IDs
    List<NotBetweenClause> clauses; // a list of conditions the solution must abide by

    class NotBetweenClause {
        int boundary1Id, boundary2Id, bannedId;

        public NotBetweenClause(int boundary1Id, int boundary2Id, int bannedId) {
            this.boundary1Id = boundary1Id;
            this.boundary2Id = boundary2Id;
            this.bannedId = bannedId;
        }
    }

    /**
     * Create NotBetweenProblem instance from input file
     * input file specs are explained in final project spec.
     * @param filename
     */
    public NotBetweenProblem(String filename) {
        nameId = new HashMap<>();
        clauses = new ArrayList<>();
        try (BufferedReader br = new BufferedReader(new FileReader(filename))) {
            // get number of wizards
            String numWizardsRaw = br.readLine();
            numWizards = Integer.parseInt(numWizardsRaw.trim());

            // get wizard names (and build nameId and names)
            /*
            String namesRaw = br.readLine();
            names = namesRaw.split(" ");
            for (int i = 0; i < names.length; i++) {
                nameId.put(names[i], i);
```

```
            }
            */
            int nameIndex = 0;
            names = new String[numWizards];

            // get number of clauses
            String numClausesRaw = br.readLine();
            int numClauses = Integer.parseInt(numClausesRaw.trim());


            // build clauses
            String clauseRaw = null;
            while ((clauseRaw = br.readLine()) != null) {
                String[] clauseTerms = clauseRaw.split(" ");
                // add new names to the list
                for (String name : clauseTerms) {
                    if (!nameId.containsKey(name)) {
                        names[nameIndex] = name;
                        nameId.put(name, nameIndex);
                        nameIndex++;
                    }
                }
                // add the clause
                int boundary1Id = nameId.get(clauseTerms[0]);
                int boundary2Id = nameId.get(clauseTerms[1]);
                int bannedId   = nameId.get(clauseTerms[2]);
                clauses.add(new NotBetweenClause(boundary1Id, boundary2Id, bannedId));
            }
        } catch (IOException e) {
            System.out.println("Error reading file. Exiting.");
            exit(1);
        }
    }

    /**
     * Given a list of wizard IDs, return the corresponding list of names
     * @param ids
     * @return
     */
    public String[] convertToNames(List<Integer> ids) {
        String[] result = new String[ids.size()];
        int index = 0;
        for (int id : ids) {
            result[index] = names[id];
            index++;
        }
        return result;
    }
}
```

4.4 Class: Satisfiability Problem

```
import org.sat4j.core.VecInt;
import org.sat4j.minisat.SolverFactory;
import org.sat4j.specs.ContradictionException;
import org.sat4j.specs.IProblem;
import org.sat4j.specs.ISolver;
import org.sat4j.specs.TimeoutException;

import java.util.*;
```

```java
/**
 * Created by eamonncarson on 11/10/17.
 */
public class SatisfiabilityProblem {

    class Relation {
        // a < b
        int a, b;
        boolean flipped = false;

        public Relation(int a, int b) {
            if (a > b) {
                int tmp = a;
                a = b;
                b = tmp;
                flipped = true;
            }
            this.a = a;
            this.b = b;
        }

        @Override
        public int hashCode() {
            return a * numObjects + b;
        }

        @Override
        public boolean equals(Object obj) {
            if (obj instanceof Relation) {
                Relation that = (Relation) obj;
                return (this.a == that.a) && (this.b == that.b);
            } else {
                return false;
            }
        }
    }

    class idComparator implements Comparator<Integer> {
        @Override
        public int compare(Integer o1, Integer o2) {
            Relation r = new Relation(o1, o2);
            int rid = relationId.get(r);
            // note that rids are 1 indexed
            int relationHolds = solution[rid - 1];
            if (r.flipped) {
                relationHolds = -relationHolds;
            }
            return relationHolds;
        }
    }

    private int numObjects;
    private int numRelations;
    private Relation[] relations;
    private Map<Relation, Integer> relationId;
    private int[] solution;

    /**
     * Generate a SAT problem that corresponds the the
     * NotBetweenProblem given, and solve it.
     * @param nbp
```

```
 */
public SatisfiabilityProblem(NotBetweenProblem nbp) throws ContradictionException,
    TimeoutException {
    generateRelations(nbp);
    ISolver solver = SolverFactory.newDefault();
    solver.newVar(numRelations);
    // Add nbp clauses
    for (NotBetweenProblem.NotBetweenClause nbpClause : nbp.clauses) {
        // get canonical forms of relations
        Relation relation1 = new Relation(nbpClause.boundary1Id, nbpClause.bannedId);
        Relation relation2 = new Relation(nbpClause.bannedId, nbpClause.boundary2Id);
        // look up their ids
        Integer testId1 = relationId.get(relation1);
        Integer testId2 = relationId.get(relation2);
        // If the same name appears twice, the clause is meaningless. Ignore.
        if (testId1 == null || testId2 == null) continue;
        int relation1Id = testId1;
        int relation2Id = testId2;
        // see if they are the inverted versions
        relation1Id = relation1.flipped ? -relation1Id : relation1Id;
        relation2Id = relation2.flipped ? -relation2Id : relation2Id;
        // add to the SAT problem
        int[] literals = {relation1Id, relation2Id};
        solver.addExactly(new VecInt(literals), 1);
    }
    // Add relation consistency clauses
    for (int upper = 0; upper < numObjects; upper++) {
        for (int lower = 0; lower < upper; lower++) {
            for (int inter = lower + 1; inter < upper; inter++) {
                Relation relation1 = new Relation(lower, inter);
                Relation relation2 = new Relation(inter, upper);
                Relation relation3 = new Relation(lower, upper);
                int id1 = relationId.get(relation1);
                int id2 = relationId.get(relation2);
                int id3 = relationId.get(relation3);
                int[] literals1 = {-id1, -id2, id3}; // id1 and id2 ==> id3
                int[] literals2 = {id1, id2, -id3}; // -id1 and -id2 ==> -id3
                solver.addClause(new VecInt(literals1));
                solver.addClause(new VecInt(literals2));
            }
        }
    }
    // Solve
    IProblem problem = solver;
    solution = problem.findModel();
}

/**
 * Return a list of ids which fulfill the conditions of the nbp
 * @return
 */
public List<Integer> getSolution() {
    // initialize array 0 ... numObjects
    ArrayList<Integer> order = new ArrayList<>();
    for (int i = 0; i < numObjects; i++) {
        order.add(i);
    }
    // sort it using the relation booleans we know
    order.sort(new idComparator());
    return order;
}
```

```
    public List<Integer> getSolution2() {
        int[] numGreater = new int[numObjects];
        for (int id = 0; id < numObjects; id++) {
            for (int otherId = 0; otherId < numObjects; otherId++) {
                if (otherId != id) {
                    Relation r = new Relation(id, otherId);
                    int rid = relationId.get(r);
                    int truth = r.flipped ? -solution[rid - 1] : solution[rid - 1];
                    if (truth > 0) { //the relation is true, so id < otherId
                        numGreater[id]++;
                    }
                }
            }
        }
        ArrayList<Integer> numGreaterList = new ArrayList<>();
        for (int elem : numGreater) {
            numGreaterList.add(elem);
        }
        ArrayList<Integer> order = new ArrayList<>();
        for (int i = 0; i < numObjects; i++) {
            order.add(numGreaterList.indexOf(i));
        }
        return order;
    }

    /**
     * generate the set of relation boolean inputs for the SAT problem
     * @param nbp
     */
    private void generateRelations(NotBetweenProblem nbp) {
        numObjects = nbp.numWizards;
        numRelations = numObjects * (numObjects - 1) / 2;
        relations = new Relation[numRelations + 1];
        relationId = new HashMap<>(numRelations);
        int index = 1;
        for (int i = 0; i < numObjects; i++) {
            for (int j = 0; j < i; j++) {
                Relation relation = new Relation(j, i);
                relations[index] = relation;
                relationId.put(relation, index);
                index++;
            }
        }
    }
}
```