Notes about transforming Rikudo to SAT

Gaëtan REGAUD Chanattan SOK
Jules TIMMERMAN

February 2023

Introduction

In this program, we are transforming a Rikudo instance in a SAT problem.

The Rikudo problem will be represented as a graph $G = (V, E)$, with a finite number of vertices.

To do that we will translate the Rikudo problem using $(x_{i,v})_{\substack{i\in[\![1,n]\!]\\ v\in[\![1,n]\!]}}$ where n is the number of nodes. Intuitively, these variables will represent whether the vertex v is labeled as i in the computed path.

Note that in the program itself, we will mostly use indices in the range $[\![0, n-1]\!]$

Encoding the game's logic

Each vertex appears once

To encode the fact that each vertex v can appear only once in the path, we will first encode that the vertex appear at least once, and then at most once :

$$\Phi_v = \left(\bigvee_{i=1}^{n} x_{i,v}\right) \wedge \left(\bigwedge_{i=1}^{n}\bigwedge_{\substack{j=1\\ j\neq i}}^{n} (x_{i,v} \Rightarrow \neg x_{j,v})\right)$$

Because $p \Rightarrow q \equiv \neg p \vee q$, we directly have the formula in CNF :

$$\Phi_v = \left(\bigvee_{i=1}^{n} x_{i,v}\right) \wedge \left(\bigwedge_{i=1}^{n}\bigwedge_{\substack{j=1\\ j\neq i}}^{n} (\neg x_{i,v} \vee \neg x_{j,v})\right)$$

Each labeled is given once

We encode that the same way we encoded the vertices, for $i \in [\![1, n]\!]$:

$$\Phi'_v = \left(\bigvee_{v=1}^{n} x_{i,v} \right) \wedge \left(\bigwedge_{v=1}^{n} \bigwedge_{\substack{w=1 \\ w \neq v}}^{n} (\neg x_{i,v} \vee \neg x_{i,w}) \right)$$

Consecutively labeled vertices are adjacent

We need to have that the vertex v has at least one neighbor with an incremented label :

$$\Psi_v = \bigwedge_{i=1}^{n-1} \left(x_{i,v} \Rightarrow \bigvee_{\substack{w=1 \\ (w,v) \in E}}^{n} x_{i+1,w} \right)$$

Then the same way as before, we already have a CNF :

$$\Psi_v = \bigwedge_{i=1}^{n-1} \left(\neg x_{i,v} \vee \bigvee_{\substack{w=1 \\ (w,v) \in E}}^{n} x_{i+1,w} \right)$$

Some vertices are fixed

In the Rikudo game, some vertices (other than start and finish) will have a fixed label as a constraint. We can simply add the constant representing it. That is, if, for every fixed vertex v, we let $f(v)$ be the fixed label it has, we have :

$$\psi_v = x_{f(v),v}$$

Diamonds

To encode the diamonds, we will consider v and w, two vertices that are separated by a diamond. If v has a certain label i, then w is either labeled $i+1$ or $i-1$. It gives us :

$$\Psi'_{v,w} = \bigwedge_{i=2}^{n-1} [(x_{i,v} \Rightarrow (x_{i+1,w} \vee x_{i-1,w})) \wedge (x_{i,w} \Rightarrow (x_{i+1,v} \vee x_{i-1,v}))]$$

Final logical formula

We need to have all the above formulas true, so the final logical proposition is (without diamonds) :

$$\mathcal{P} = \bigwedge_{v=1}^{n} \Phi_v \bigwedge_{i=1}^{n} \Phi'_i \bigwedge_{v=1}^{n} \Psi_v \bigwedge_{v \text{ fixed}} \psi_v$$

If we want to take in consideration the diamonds, we instead have :

$$\mathcal{P}' = \mathcal{P} \wedge \bigwedge_{v=0}^{n-1} \bigwedge_{\substack{w=0 \\ v,w \text{ separated} \\ \text{by a diamond}}}^{n-1} \Psi'_{v,w}$$

Remark

- We can see that the transformation is polynomial.

Using a SAT-solving library

The library

As we were using Java, we decided to use the Sat4j library. It can solve different type of problems, but we will only use it as a SAT-solver.

To work with it, we need to give it a logical formula in the Dimacs CNF format. Basically, the Dimacs CNF format is a way to encode a logical formula using integers. Each variable will be represented by an integer strictly greater than 0. If its negation appear in a clause, it will be negative. In the Dimacs CNF format, we separate every clause of the CNF by a 0. Here, we won't have to do that as we will add each clause separately.

Our usage

We will represent CNF with `ArrayList<ArrayList<Integer>>` objects, where the inner list is a clause.

We need to supply the solver with a CNF formula, so we need to convert our formula to CNF. Many of our subformulas are in DNF form, so we are converting these directly to DNF and then concatenating them in one big CNF.

Mapping variables to integers

Our variables are written $x_{i,v}$ for $i, v \in [\![1, n]\!]^2$. We need to map those to integers in $\mathbb{N}^*$ as 0 is used to separated clauses.

We will use the "usual" bijection :

$$f : \begin{cases} \mathbb{N} \times \mathbb{N} & \to \mathbb{N}^* \\ i, v & \mapsto \frac{(i+v)(i+v+1)}{2} + v + 1 \end{cases}$$

The idea behind that function is to map i, v to a diagonal where $i + v$ is a constant. We then add v to order the diagonal.

For the inverse function, the idea is to compute k the diagonal by repeatedly subtracting the previous diagonal.