Project Report - CMPT 417

Luiz Fernando Peres de Oliveira - 301288301 - lperesde@sfu.ca

I. Introduction

This project aims to solve the **Pizza** problem given in the *LP/CP Programming Contest 2015*, where some students in the University College Cork want to make a large order of pizza for a party such that they use as many vouchers collected throughout the year as they can. The final objective is to use the vouchers so to obtain all ordered pizzas with the least possible cost.

Because we want to minimize the total possible cost for all ordered pizzas, the problem is then an optimization problem and not only satisfiability, meaning that we start our initial K, where K is our target cost, with the sum of prices of all pizzas with no vouchers used, as this is the maximum possible cost for this problem and, therefore, if we were to choose any K larger than that, K could be improved to at least as good as the total sum of the pizza prices. We repeat the process by looking for satisfiable instances of that K' that are smaller than our previous K and stop whenever we cannot satisfy K', meaning that K' is the least possible value of K.

In this document, we first specify the problem by defining our vocabulary $\mathcal{L}$ of functions and constant symbols for an $\mathcal{L}$-structure $\mathcal{M}$. Then we will show that, to solve this problem, we must find a vocabulary $\mathcal{L}'$ (with a $\mathcal{L}'$-structure $\mathcal{M}'$) that extends $\mathcal{L}$ such that when we find a satisfiable instance of M for all constraints in $\mathcal{L}'$, we find a satisfying instance for the initial problem, since $\mathcal{M}'$ and $\mathcal{M}$ share the same universe M. After specifying our vocabularies and constraint relations, we will show some tests instances of the problem run on **Minizinc** solver that will be then have their performance empirically evaluated. Finally, we will have a short discussion about the process of solving the Pizza problem.

II. Specification

The pizza problem is as follows:

- The final goal is to obtain all ordered pizzas for the least possible cost.
- A voucher is a pair of numbers (a, b) where you pay for a pizzas and obtain b pizzas for free as long as each of the b pizzas cost no more than each of the a pizzas.
- A voucher does not need to be completely used and not all vouchers need to be used.

A vocabulary $\mathcal{L}$ then can be defined by the symbols $[price, buy, free, n, m, k]$ where n is the number of pizzas, m is the number of coupons, k is the cost bound, $price : [n] \to \mathbb{N}$ is an unary function that maps the price of each one of the n pizzas, $buy : [m] \to \mathbb{N}$ is an unary function that maps the number of pizzas that must be bought for each one of the m vouchers and $free : [m] \to \mathbb{N}$ is an unary function that maps the number of pizzas that come for free when a voucher is used, for each one of the m vouchers. The universe M are all the numbers appearing in the structure. The **Minizinc** equivalent form of $\mathcal{L}$ is:

int: n;
array[1..n] **of int**: price;
int: m;
array[1..m] **of int**: buy;
array[1..m] **of int**: free;

We need to find an assignment of pizzas and coupons that minimize the total cost k and for this reason we will create a vocabulary $\mathcal{L}'$ that extends $\mathcal{L}$, where $\mathcal{L}$ and $\mathcal{L}'$ share the same universe. Let $\mathcal{L}'$ be defined by the symbols in $\mathcal{L}$ and the symbols $[Paid, Used, Justifies, UsedFor]$, where the symbol $Paid : [n] \to \{0, 1\}$ is an unary symbol representing the set of paid pizzas, $Used : [m] \to \{0, 1\}$ is an unary symbol representing the set of used coupons, $Justifies : [m] \to [n] \to \{0, 1\}$ is a binary symbol representing the set of coupons c that will be used by paying for pizzas p and $UsedFor : [m] \to [n] \to \{0, 1\}$ is a binary symbol representing the set of free pizzas p that were obtained by using coupons c. The **Minizinc** equivalent form of $\mathcal{L}'$ is:

array[1..n] **of var bool**: Paid;
array[1..m] **of var bool**: Used;
array[1..m, 1..n] **of var bool**: Justifies;
array[1..m, 1..n] **of var bool**: UsedFor;

III. Testing

IV. Empirical Performance

V. Discussion

VI. Data

References

[1] "Function Types", Bartosz Milewski's Programming Cafe, 2017. [Online]. Available: https://bartoszmilewski.com/2015/03/13/function-types/. [Accessed: 29- Nov- 2017].