Report of the final project for the constraint programming course

Xiaoke(Jimmy) Shen

May 27, 2017

1 Introduction

Picat is a simple, and yet powerful, logic-based multi-paradigm programming language aimed for general-purpose applications. Picat is a rule-based language, in which predicates, functions, and actors are defined with pattern-matching rules. Picat incorporates many declarative language features for better productivity of software development, including explicit non-determinism, explicit unification, functions, list comprehensions, constraints, and tabling[1]. For the constraints, three solvers are provided: cp, sat and mip. In this article, the cp solver and the sat solver are used to resolve the stock strategy problem. This problem is described in section 2 and the results are also shown in sections. In section 3, the conclusion is given.

2 The stock Problem

2.1 Brief into of the problem

In the Enterprise Business market, as the customers always have a very strict requirement of the lead time(product delivery time), the stock of the product is critical for the business success of the reseller. If part of the products in the order are not in the stock, the reseller will lose the order. However, as the limited cash flow, the reseller can not stock all the products needed by the customer which means the reseller will lose some order as short of stock. The reseller has to optimize the stock strategy to maximize the profit.

2.2 Problem Description

The reseller buy products from the vendor with a "Buying Price from vendor" and then sell the product to the end customer by adding a profit margin. Say the buying price of product i is $P_{(i)}^{buy}$ and the selling price is $P_{(i)}^{sell}$, then the profit margin of this product is :

$$Margin_{(i)} = \frac{P_{(i)}^{sell} - P_{(i)}^{buy}}{P_{(i)}^{buy}}$$

As the lead time issue mentioned in the first section, the reseller has to stock the product to win the order. If the reseller doesn't have enough stock, then the reseller will lose the order. Assume the reseller has only C_1 cash flow to buy the stock product every month and the reseller knows well about the orders he will have in the coming month. This problem will be changed to a constraint problem and it can be formally described as below:

$$\max Profit_{total} = \sum_{j \in won \quad orders} Profit_{(j)}$$

, where

$$Profit_{(j)} = \sum_{i} P_{(i)}^{buy} * Margin_{(i)} * N_{(i)}^{order}$$

$N_{(i)}^{order}$ is the number of product i in that order.
constraint to :

$$\sum_{i} P_{(i)}^{buy} * N_{(i)}^{stock} \leq C_1$$

$N_{(i)}^{stock}$ is the number of stock product i, C_1 is the cash flow can be used in one month for the stock product purchasing.

2.3 A Specified Problem

Say one reseller can resell 8 products to the market and the info of these 8 products are given in Figure 1. And the reseller has 20 orders in the coming one month. The detail info of the order is given in figure 2. The cash flow C_1 is approximated to 40% of the total value of these 20 orders. How can the reseller to make the stock strategy to earn the maximum profit?

Product Name	Product NO.	Unit Buying Price from vendor	Profit margin
High capacity Router	1	8000	30%
Low capacity Router	2	4000	15%
High capacity Switch	3	6000	25%
Low capacity Switch	4	1000	10%
Storage	5	5000	40%
Server	6	10000	45%
WLAN	7	600	50%
Telepresence Terminal	8	2000	40%

Figure 1: Unit buying price and profit margin of each product

Order NO.	High capacity Router	Low capacity Router	High capacity Switch	Low capacity Switch	Storage	Server	WLAN	Telepresence Terminal
1	1	3	1	4	0	1	8	0
2	0	0	0	0	3	6	0	0
3	0	0	0	0	0	0	0	3
4	3	6	1	4	0	2	8	0
5	1	3	1	4	0	1	0	3
6	2	3	2	8	0	2	8	0
7	0	0	0	0	4	6	0	0
8	0	0	0	0	0	0	0	3
9	0	0	0	0	3	8	0	0
10	2	3	1	4	0	1	8	0
11	2	0	0	0	0	0	0	0
12	3	6	1	4	0	2	8	0
13	3	3	1	4	0	1	0	3
14	2	3	2	8	0	2	8	0
15	0	0	0	0	4	6	0	0
16	1	5	1	6	0	0	0	3
17	3	6	1	4	0	2	8	0
18	1	3	1	3	0	1	0	3
19	2	3	2	9	0	2	8	0
20	0	0	1	5	4	6	0	0

Figure 2: 20 orders with detail info in the coming month

2.4 Solution and result for this specified problem

This is a CSP problem and this specified problem(as described in the previous session) can be well resolved by using the Picat. Both cp and sat can well resolve this specified problem.
The code of the cp version is given in the file: project_stock_problem_easy_cp.pi
The output of the cp version is given in the file: project_stock_problem_easy_cp_output.txt
The code of the sat version is given in the file: project_stock_problem_easy_sat.pi
The output of the sat version is given in the file: project_stock_problem_easy_sat_output.txt

The result comparison for different solvers to the specified problem is given in table 1.

Table 1: **Result comparison** for different solvers to the specified problem

Problem	Solver	Result	time(seconds)
the specified problem	cp	success	5.222
the specified problem	sat	success	3.512

2.5 Solution and the result for the hard one

In the specified problem(as described in the section 2.3), we only have 20 sales orders. In order to make the program workable for more order scenario, a hard version is generated in this session. The product number for each order is generated by a random number from 0 to 9 and the total number of the order is set to 20, 50, 100 and 200 to test the performance.

Two different algorithms are used to resolve this simulated stock strategy problem. The first one is using the similar algorithm used for the specified problem. The basic idea is directly using a declarative provided by Picat to describe the problem and try to get the solution directly. It works when the problem size is small for example 20. However, when the problem size is bigger, such as 100, it can not be well resolved by using the Picat resolver directly. A modified algorithm is used in the version beta. The algorithm used for the beta version is described as below:

Data: SalesOpportunity, CashFlow
Result: Give the stock strategy to get the maximum Profits.
random generate N orders;
Sort those N orders by the average profit margin ratio;
M = 1;
while *Have unconsidered sales opportunities and the maximum profit can still be increased* **do**
    Find the storage strategy from the first M orders of the sorted order list;
    M = M +1;
end

Algorithm 1: Algorithm used in the beta version

The result comparison for different solvers and different code versions to the harder version of the stock strategy problem is given in table 2.
The code and the output file info is provided in table 3 and table 4.

Table 2: **Result comparison** for different solvers and different code versions to the harder version of the stock strategy problem

ProblemSize	CodeV.	Solver	Result	time(s)
20	original	cp	success	6.617
50	original	cp	N/A	cannot get the output after a long time waiting
100	original	cp	N/A	cannot get the output after a long time waiting
200	original	cp	N/A	cannot get the output after a long time waiting
20	original	sat	success	28.446
50	original	sat	N/A	can not get the output after a long time waiting
100	original	sat	N/A	can not get the output after a long time waiting
200	original	sat	N/A	can not get the output after a long time waiting
20	beta	cp	success	1.814
50	beta	cp	success	40.691
100	beta	cp	success	47.694
200	beta	cp	success	297.828
20	beta	sat	fail	can not get the correct answer
50	beta	sat	fail	can not get the correct answer
100	beta	sat	fail	can not get the correct answer
200	beta	sat	fail	can not get the correct answer

Table 3: Code file name related to different size, different versions and different solvers

ProblemSize	CodeV.	Solver	Code file name
20	original	cp	project_stock_problem_hard_cp_20.pi
50	original	cp	project_stock_problem_hard_cp_50.pi
100	original	cp	project_stock_problem_hard_cp_100.pi
200	original	cp	project_stock_problem_hard_cp_200.pi
20	original	sat	project_stock_problem_hard_sat_20.pi
50	original	sat	project_stock_problem_hard_sat_50.pi
100	original	sat	project_stock_problem_hard_sat_100.pi
200	original	sat	project_stock_problem_hard_sat_200.pi
20	beta	cp	project_stock_problem_hard_cp_beta_20.pi
50	beta	cp	project_stock_problem_hard_cp_beta_50.pi
100	beta	cp	project_stock_problem_hard_cp_beta_100.pi
200	beta	cp	project_stock_problem_hard_cp_beta_200.pi
20	beta	sat	project_stock_problem_hard_sat_beta_20.pi
50	beta	sat	project_stock_problem_hard_sat_beta_50.pi
100	beta	sat	project_stock_problem_hard_sat_beta_100.pi
200	beta	sat	project_stock_problem_hard_sat_beta_20.pi

3 Conclusion

The Picat can well resolve the constraint problem in an efficient way. Meanwhile, when the problem size is big, the combination of the improved algorithm and the constraint solvers provided by Picat will achieve a good performance.

References

[1] *http://picat-lang.org/*

Table 4: Output file name related to different size, different versions and different solvers

ProblemSize	CodeV.	Solver	Code file name
20	original	cp	project_stock_problem_hard_cp_20_output.txt
50	original	cp	project_stock_problem_hard_cp_50_output.txt
100	original	cp	project_stock_problem_hard_cp_100_output.txt
200	original	cp	project_stock_problem_hard_cp_200_output.txt
20	original	sat	project_stock_problem_hard_sat_20_output.txt
50	original	sat	project_stock_problem_hard_sat_50_output.txt
100	original	sat	project_stock_problem_hard_sat_100_output.txt
200	original	sat	project_stock_problem_hard_sat_200_output.txt
20	beta	cp	project_stock_problem_hard_cp_beta_20_output.txt
50	beta	cp	project_stock_problem_hard_cp_beta_50_output.txt
100	beta	cp	project_stock_problem_hard_cp_beta_100_output.txt
200	beta	cp	project_stock_problem_hard_cp_beta_200_output.txt
20	beta	sat	project_stock_problem_hard_sat_beta_20_output.txt
50	beta	sat	project_stock_problem_hard_sat_beta_50_output.txt
100	beta	sat	project_stock_problem_hard_sat_beta_100_output.txt
200	beta	sat	project_stock_problem_hard_sat_beta_20_output.txt