DECISION ANALYTICS

ASSIGNMENT 1- CONSTRAINT PROGRAMMING

1. There are five houses, each of a different colour and inhabited by men of different nationalities, with different pets, drinks, and cigarettes. Who owns the cat?

Variables and domain:

n, c, p, d, cig {0, 1, 2, 3, 4}

{English, Spaniard, Ukrainian, Norwegian, Japanese} == {0, 1, 2, 3, 4}

{Red, Green, Ivory, Blue, Yellow} == {0, 1, 2, 3, 4}

{Dog, Snail, Fox, Horse, Cat} == {0, 1, 2, 3, 4}

{Coffee, Tea, Milk, Orange juice, Unknown} == {0, 1, 2, 3, 4}

{Old Gold, Kools, Chester Fields, Parliaments, Lucy Striker} == {0, 1, 2, 3, 4}

Constraints:

1. The Englishman lives in the red house.

English == Red

1. The Spaniard owns the dog.

Spaniard == Dog

1. Coffee is drunk in the green house.

Coffee == Green

1. The Ukrainian drinks tea.

Ukrainian == Tea

1. The green house is immediately to the right of the ivory house.

Green == (Ivory + 1)

1. The Old Gold smoker owns snails.

OldGold == Snail

1. Kools are smoked in the yellow house.

Kools == Yellow

1. Milk is drunk in the middle house.

Milk == 2

1. The Norwegian lives in the first house on the left.

Norwegian == 0

1. The man who smokes Chesterfields lives in the house next to the man with the fox.

abs(Chesterfields - Fox) == 1

1. Kools are smoked in the house next to the house where the horse is kept.

abs(Kools - Horse) == 1

1. The Lucky Strike smoker drinks orange juice.

Lucystrike == Orangejuice

1. The Japanese smoke Parliaments.

Japan == Parliaments

1. The Norwegian lives next to the blue house.

abs(Norway - Blue) == 1

Heuristics:

The search method used is the Depth First Search. In DFS, every possible combinations of variables. The execution starts at a root node. There are two steps at every node, first there is constraint propagation to reduce the domains and second is branching. In branching, the algorithm starts with smallest domain first and then assigns the smallest value in the domain[1].

The search stops in a leaf node when there is no possibility of branching or if one domain is emptied, which means this branch cannot give a possible solution.

Solution:

The solution for the problem is: ‘Japanese owns the cat’. The program prints the output with this solution.

1. A company shall invest in some building projects, so that it is obtaining the maximum total value from them.

Variables and domain:

Projects {0, 1, … 14}

output variable: final\_projects {0, 1}

The output variable is a Boolean array that contains the selected projects with the value corresponding to index of selected projects taking value 1 and that of not-selected projects taking values 0.

The value, budget, no. of staff needed, required projects and not-with projects are all given as constant inputs in the code. The Value of project is used for optimisation. The objective of code is to maximise the total value of projects selected.

Constraints:

1. budget of 225 M€ (million €)

total\_budget <= max\_budget

1. 28 staff available

total\_staff <= max\_staff

1. maximum 9 projects can be selected

total\_projects <= max\_no\_proj

1. some project must not be selected together with other projects, and some projects must be selected together with other

Projects not to be selected with other projects

List: [1 - 10, 5 - 6, 11 - 15]

In this case, if project 1 is selected, Project 10 must not be selected. Hence, in the output array, if Project 1 has value 1 then, Project 10 will have value 0 and like wise the same with projects 5 and 6 and the same with projects 11 and 15.

Project that require some projects

List: [3- 15, 4- 15, 8- 7, 13- 2, 14- 2]

In this case, if project 3 s selected, project 15 should also be selected. Hence, in the output array, if project 3 has value 1 then Project 15 should also have value 1 in the output array. This is the same with projects

Heuristics:

The algorithm used for optimization of an objective function is Branch and Bound search. During branch and bound search, there are two steps. Partitioning the search space is the first step and pruning the non-optimal solutions is the second and the algorithm progresses [1].

The maximum total value obtained for the problem is **2370**. The output prints the combination of projects which satisfies this maximum total value. The program can also print all the possible solutions which satisfy all the given conditions but with values less than maximum values. All the solutions are attached towards the end of this document.

**Implementations of the Program for the Ex: 1- who owns the cat:**

#include <gecode/int.hh>

#include <gecode/search.hh>

#include <gecode/minimodel.hh>

using namespace Gecode;

using namespace std;

class Houses : public Space {

protected:

// Array initialisation for Nationality

IntVarArray n;

// Array initialisation for Color of house

IntVarArray c;

// Array initialisation for Pets

IntVarArray p;

// Array initialisation for Drinks

IntVarArray d;

// Array initialisation for Cigarette

IntVarArray cig;

public:

Houses(void) : n(\*this, 5, 0, 4), c(\*this, 5, 0, 4), p(\*this, 5, 0, 4), d(\*this, 5, 0, 4),

cig(\*this, 5, 0, 4)

{

//Initialising value\_index to values in the integer arrays

IntVar English(n[0]), Spaniard(n[1]), Ukranian(n[2]), Norway(n[3]), Japan(n[4]);

IntVar Red(c[0]), Green(c[1]), Ivory(c[2]), Blue(c[3]), Yellow(c[4]);

IntVar Dog(p[0]), Snail(p[1]), Fox(p[2]), Horse(p[3]), Cat(p[4]);

IntVar Coffee(d[0]), Tea(d[1]), Milk(d[2]), Orangejuice(d[3]), unk(d[4]);

IntVar OldGold(cig[0]), Kools(cig[1]), Chesterfields(cig[2]), Parliaments(cig[3]), Lucystrike(cig[4]);

// Constraint that each of nationalities, house colour, pets, drinks and cigarettes

// should take distinct values

distinct(\*this, n);

distinct(\*this, c);

distinct(\*this, p);

distinct(\*this, d);

distinct(\*this, cig);

//constraints specified in the problem

////The Englishman lives in the red house

rel(\*this, English == Red);

////The Spaniard owns the dog

rel(\*this, Spaniard == Dog);

////Coffee is drunk in the green house

rel(\*this, Coffee == Green);

////The Ukrainian drinks tea

rel(\*this, Ukranian == Tea);

//// The green house is immediately to the right of the ivory house

rel(\*this, Green == (Ivory + 1));

////The Old Gold smoker owns snails

rel(\*this, OldGold == Snail);

////Kools are smoked in the yellow house

rel(\*this, Kools == Yellow);

////Milk is drunk in the middle house

rel(\*this, Milk == 2);

////The Norwegian lives in the first house on the left

rel(\*this, Norway == 0);

////The man who smokes Chesterfields lives in the house next to the man with the fox

rel(\*this, abs(Chesterfields - Fox) == 1);

////Kools are smoked in the house next to the house where the horse is kept

rel(\*this, abs(Kools - Horse) == 1);

////The Lucky Strike smoker drinks orange juice

rel(\*this, Lucystrike == Orangejuice);

////The Japanese smoke Parliaments

rel(\*this, Japan == Parliaments);

////The Norwegian lives next to the blue house.

rel(\*this, abs(Norway - Blue) == 1);

//Branching the propagation

branch(\*this, n, INT\_VAR\_SIZE\_MIN(), INT\_VAL\_MIN());

branch(\*this, c, INT\_VAR\_SIZE\_MIN(), INT\_VAL\_MIN());

branch(\*this, p, INT\_VAR\_SIZE\_MIN(), INT\_VAL\_MIN());

branch(\*this, d, INT\_VAR\_SIZE\_MIN(), INT\_VAL\_MIN());

branch(\*this, cig, INT\_VAR\_SIZE\_MIN(), INT\_VAL\_MIN());

}

// search support for all five variable arrays

Houses (bool share, Houses& s) : Space(share, s)

{

n.update(\*this, share, s.n);

c.update(\*this, share, s.c);

p.update(\*this, share, s.p);

d.update(\*this, share, s.d);

cig.update(\*this, share, s.cig);

}

virtual Space\* copy(bool share) {

return new Houses (share, \*this);

}

// print solution

void print(void) {

cout << "Nationality : "<<n << endl;

cout << "Color of House: "<< c << endl;

cout << "Pet : "<<p << endl;

cout << "Drink : "<< d << endl;

cout << "Cigarette : "<< cig ;

cout << " " << endl;

string Nationalities[5] = {"English", "Spaniard", "Ukranian", "Norwegian", "Japanese"};

//cat is the variable. It is assigned as the 4th value in the array of pets

int cat\_index = p[4].val();

int N\_index;

cout << "Index Value for Cat: "<< cat\_index << endl; ;

for(int i = 0; i < 5; i++)

{

if(cat\_index == n[i].val())

N\_index = i;

}

cout << Nationalities[N\_index] << " owns the cat";

}

};

// main function

int main(int argc, char \*argv[]) {

// create model and search engine

Houses\* h = new Houses();

DFS<Houses> e(h);

delete h;

// search and print all solutions

if (Houses\* s = e.next()) {

s->print();

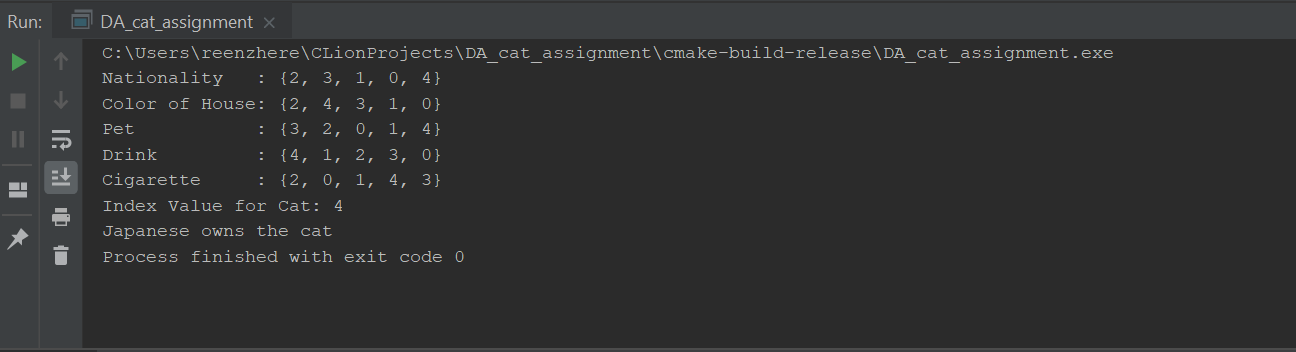
delete s;

}

return 0;

}

Output of the above code:



**Implementations for Ex.2 – Building Project**

#include <gecode/int.hh>

#include <gecode/search.hh>

#include <gecode/minimodel.hh>

using namespace Gecode;

using namespace std;

class BuildingProject : public Space{

protected:

static const int proj = 15;

static const int max\_no\_proj = 9;

static const int max\_budget = 225;

static const int max\_staff = 28;

//the output arrays

//a boolean array to store the selected projects

IntVarArray final\_projects;

//Variable to store the final number for each of projects, budget and staff

IntVar total\_projects;

IntVar total\_budget;

IntVar total\_staff;

//Variable to hold the objective function

IntVar total\_value;

public:

BuildingProject():

final\_projects(\*this, proj, 0, 1),

total\_projects(\*this, 0, max\_no\_proj),

total\_budget(\*this, 0, max\_budget),

total\_staff(\*this, 0, max\_staff),

total\_value(\*this, 0, 10000)

{

//Value corresponding to each project given as input to the problem

const int Value[15] = {600, 400, 100, 150, 80, 120, 200, 220, 90, 380, 290, 130, 80, 270, 280};

const int Budget[15] = {35, 34, 26, 12, 10, 18, 32, 11, 10, 22, 27, 18, 16, 29, 22};

const int no\_of\_staff[15] = {5, 3, 4, 2, 2, 2, 4, 1, 1, 5, 3, 2, 2, 4, 3};

const int not\_with[12] = {1, 10, 5, 6, 6, 5, 10, 1, 11, 15, 15, 11};

const int requires[10] = {3, 15, 4, 15, 8, 7, 13, 2, 14, 2};

IntArgs proj\_values(proj, Value);

IntArgs proj\_Budget(proj, Budget);

IntArgs proj\_staff(proj, no\_of\_staff);

IntArgs proj\_requires(10, requires);

IntArgs proj\_not\_with(12, not\_with);

///constraints for each project

//distinct(\*this, final\_projects);

rel(\*this, total\_projects == sum(final\_projects));

linear(\*this, proj\_staff, final\_projects, IRT\_EQ, total\_staff);

linear(\*this, proj\_Budget, final\_projects, IRT\_EQ, total\_budget);

linear(\*this, proj\_values, final\_projects, IRT\_EQ, total\_value);

rel (\*this , total\_value >= 2370);

//maximum 9 projects can be selected

rel(\*this, total\_projects <= max\_no\_proj);

//budget of 225 M€ (million €)

rel(\*this, total\_budget <= max\_budget);

//28 staff available

rel(\*this, total\_staff <= max\_staff);

//projects not with other projects

//List: [1, 10, 5, 6, 6, 5, 10, 1, 11, 15, 15, 11]

for(int i=0; i < 6; i++)

{

int j1 = proj\_not\_with[i\*2]-1;

int j2 = proj\_not\_with[i\*2+1]-1;

rel(\*this, final\_projects[j1] + final\_projects[j2] <= 1);

}

//project that require some projects

//List: [3, 15, 4, 15, 8, 7, 13, 2, 14, 2];

for(int i=0; i < 5; i++)

{

int j1 = proj\_requires[i\*2]-1;

int j2 = proj\_requires[i\*2+1]-1;

rel(\*this, final\_projects[j1] - final\_projects[j2] <= 0);

}

branch(\*this, final\_projects, INT\_VAR\_MAX\_MAX(), INT\_VAL\_MAX());

}

/// Constructor for cloning \a s

// search support and update for all five variable arrays

BuildingProject (bool share, BuildingProject& s) : Space(share, s)

{

final\_projects.update(\*this, share, s.final\_projects);

total\_projects.update(\*this, share, s.total\_projects);

total\_budget.update(\*this, share, s.total\_budget);

total\_staff.update(\*this, share, s.total\_staff);

total\_value.update(\*this, share, s.total\_value);

}

/// Copy during cloning

virtual Space\* copy(bool share) {

return new BuildingProject(share,\*this);

}

/// Print solution

void print(void)

{

string project\_names[15] = { "Ishall", "sporthall", "Hotell", "Restaurant", "Kontor\_A",

"Kontor\_B", "Skola", "Dagis", "Lager", "Simhall", "Hyreshus",

"Bilverstad", "Tennishall", "Idrottsanl", "Baythamn"};

cout << "Final Selected Projects:" << final\_projects << endl;

cout << "Total Value of selected Project: " << total\_value << endl;

cout << "Total Staff for selected Projects: " << total\_staff << endl;

cout << "Total Budget of all Projects : " << total\_budget << endl;

cout << "Total number of projects: " << total\_projects << endl;

cout << "The following projects should be selected" << endl;

for (int i = 0; i < proj; i++) {

if (final\_projects[i].val () == 1) {

cout << i + 1 << " ";

int project\_index = i;

cout<<project\_names[i]<<" ";

}

}

}

};

int main(int argc, char\* argv[])

{

//create model and search engine

BuildingProject \*b = new BuildingProject();

BAB<BuildingProject> e(b);

delete b;

//search and print the result

while (BuildingProject \* s = e.next()) {

s->print();

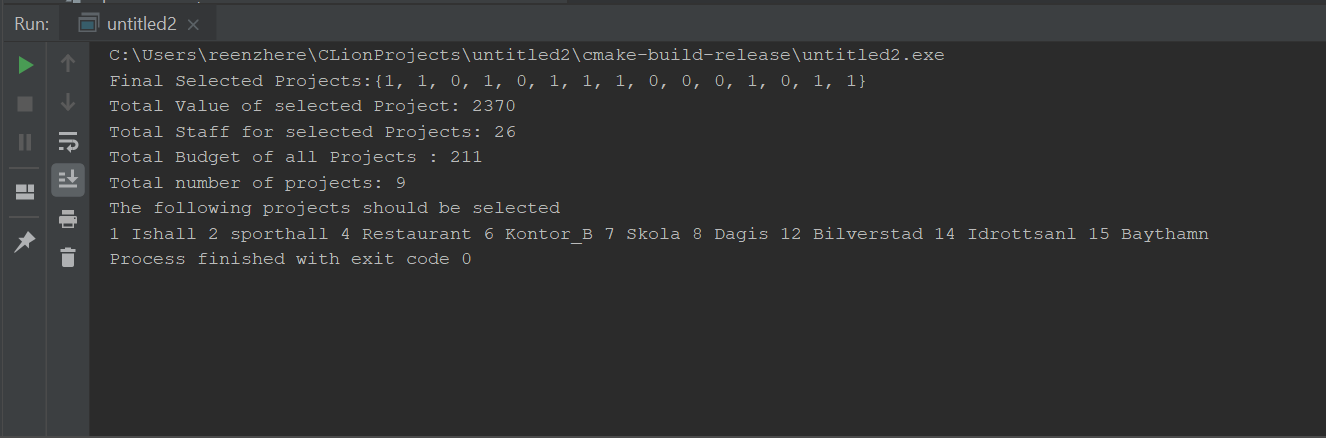
delete s;

}

return 0;

}

Output of the code with maximised value of projects:



Printing all the solutions that satisfy the constraints:

C:\Users\reenzhere\CLionProjects\untitled2\cmake-build-release\untitled2.exe

Final Selected Projects:{1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 1, 1, 1, 1}

Total Value of selected Project: 2300

Total Staff for selected Projects: 26

Total Budget of all Projects : 215

Total number of projects: 9

The following projects should be selected

1 2 6 7 8 12 13 14 15 Final Selected Projects:{1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 1, 1, 1, 1, 0}

Total Value of selected Project: 2310

Total Staff for selected Projects: 26

Total Budget of all Projects : 220

Total number of projects: 9

The following projects should be selected

1 2 6 7 8 11 12 13 14 Final Selected Projects:{1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1}

Total Value of selected Project: 2310

Total Staff for selected Projects: 25

Total Budget of all Projects : 209

Total number of projects: 9

The following projects should be selected

1 2 6 7 8 9 12 14 15 Final Selected Projects:{1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1, 0, 1, 0}

Total Value of selected Project: 2320

Total Staff for selected Projects: 25

Total Budget of all Projects : 214

Total number of projects: 9

The following projects should be selected

1 2 6 7 8 9 11 12 14 Final Selected Projects:{1, 1, 0, 1, 0, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1}

Total Value of selected Project: 2330

Total Staff for selected Projects: 26

Total Budget of all Projects : 209

Total number of projects: 9

The following projects should be selected

1 2 4 7 8 12 13 14 15 Final Selected Projects:{1, 1, 0, 1, 0, 0, 1, 1, 1, 0, 0, 1, 0, 1, 1}

Total Value of selected Project: 2340

Total Staff for selected Projects: 25

Total Budget of all Projects : 203

Total number of projects: 9

The following projects should be selected

1 2 4 7 8 9 12 14 15 Final Selected Projects:{1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1, 1}

Total Value of selected Project: 2320

Total Staff for selected Projects: 26

Total Budget of all Projects : 209

Total number of projects: 9

The following projects should be selected

1 2 4 6 7 8 13 14 15 Final Selected Projects:{1, 1, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1}

Total Value of selected Project: 2370

Total Staff for selected Projects: 26

Total Budget of all Projects : 211

Total number of projects: 9

The following projects should be selected

1 2 4 6 7 8 12 14 15 Final Selected Projects:{1, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 0, 0, 1, 1}

Total Value of selected Project: 2330

Total Staff for selected Projects: 25

Total Budget of all Projects : 203

Total number of projects: 9

The following projects should be selected

1 2 4 6 7 8 9 14 15 Final Selected Projects:{1, 1, 0, 1, 1, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1}

Total Value of selected Project: 2330

Total Staff for selected Projects: 26

Total Budget of all Projects : 203

Total number of projects: 9

The following projects should be selected

1 2 4 5 7 8 12 14 15 Final Selected Projects:{1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1}

Total Value of selected Project: 2320

Total Staff for selected Projects: 28

Total Budget of all Projects : 225

Total number of projects: 9

The following projects should be selected

1 2 3 6 7 8 12 14 15 Final Selected Projects:{1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1}

Total Value of selected Project: 2300

Total Staff for selected Projects: 28

Total Budget of all Projects : 217

Total number of projects: 9

The following projects should be selected

1 2 3 4 7 8 13 14 15 Final Selected Projects:{1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 1, 0, 1, 1}

Total Value of selected Project: 2350

Total Staff for selected Projects: 28

Total Budget of all Projects : 219

Total number of projects: 9

The following projects should be selected

1 2 3 4 7 8 12 14 15 Final Selected Projects:{1, 1, 1, 1, 0, 0, 1, 1, 1, 0, 0, 0, 0, 1, 1}

Total Value of selected Project: 2310

Total Staff for selected Projects: 27

Total Budget of all Projects : 211

Total number of projects: 9

The following projects should be selected

1 2 3 4 7 8 9 14 15 Final Selected Projects:{1, 1, 1, 1, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 1}

Total Value of selected Project: 2340

Total Staff for selected Projects: 28

Total Budget of all Projects : 219

Total number of projects: 9

The following projects should be selected

1 2 3 4 6 7 8 14 15 Final Selected Projects:{1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 0, 1, 1}

Total Value of selected Project: 2300

Total Staff for selected Projects: 28

Total Budget of all Projects : 211

Total number of projects: 9

The following projects should be selected

1 2 3 4 5 7 8 14 15

Process finished with exit code 0

REFERENCE

1. Schmidt, Benedikt. "Constraint Programming Algorithms used in Gecode."