

Chess

Due: 10:00 PM 3/7/08

“There are more adventures on a chessboard than on all the seas of the world.”

Pierre Mac Orlan, via Jose Spalenic

In modern AI, we seek to build rational agents: i.e., optimal decision makers. Such decision makers must be capable of solving optimization problems. Integer programming (and approximations thereof) can be an effective method for solving large scale optimization problems. ILOG’s CPLEX—the software package we will use to solve integer programs in this course—is quickly becoming the industry standard.

The decision-making problems we investigate in this assignment exist within the framework of the classic game of chess. The inherent mathematical nature of chess has lent mathematicians to study interesting puzzles within its confines for hundreds of years. The two puzzles we will examine are variants on the famous n -queens problem: *can n -queens be placed on a chess board such that no queen is attacking any other?*

$n \times m$ -Queens

The $n \times m$ -queens problem is played on an $n \times m$ sized rectangular chess board. Given such a board, for arbitrary n and m , your first task is to write an integer program to answer the following question: *what is the maximum number of queens that can be placed on the board such none is attacking any other?* Two sample configurations of queens on 6×9 boards appear in Figure 1—one legal and one illegal.

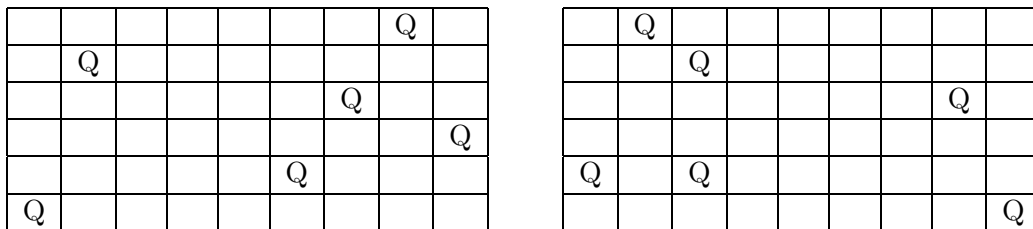


Figure 1: 6×9 -Queens. LHS: A valid configuration. RHS: An invalid configuration.

For those of you who are not familiar with the legal moves of chess pieces, a queen can move as many cells as she likes in any direction she likes—along a row, along a column, or along a diagonal. At the end of this writeup, there is a pointer to a resource with short descriptions of the legal moves in chess, which should provide all of the information about chess that is necessary to complete this assignment. Feel free to see a TA on hours if you have questions.

Read on before you start working on this problem. The second part of the problem builds on

the first, and you may want to keep that in mind while working.

$n \times m$ -MaxPoints

The second optimization problem we would you like for you solve is the following: *what is the maximum number of points that can be scored by placing queens, rooks, bishops, and knights on an $n \times m$ chess board such that none is attacking any other, where the pieces are valued at q, r, b , and k , respectively?* Typically, $q = 9$, $r = 5$, $b = 3$, and $k = 3$; however, your formulation should be general enough to handle arbitrary point values as inputs to the problem.

Integer Program

Here are some questions to think about while formulating these problems as integer programs:

- What function is being optimized (i.e. maximized or minimized)?
- What are the decision variables in this problem?
- What are the constraints on the decision variables?
e.g., Are they integer-valued or real-valued?
- How can the rules governing the movement of pieces be formulated as constraints?

What to Do

To complete this assignment, you must formulate and implement two integer programs:

1. Part I:

- Implement your formulation in Java using CPLEX and the support code, filling in the methods in the class `NMQueensSolver`.

2. Part II:

- Formulate the $n \times m$ -MaxPoints problem as an integer program.
- Implement your formulation in Java using CPLEX and the support code, filling in the methods in the class `MaxPointsSolver`.

If this is your first time writing integer programs, we recommend that you discuss your written formulations with a TA on hours before attempting to implement them. **Note:** The elegance of your formulation is worth a *significant* percentage of your final grade. Please expend an appropriate amount of effort in your design.

What to Handin

Hand in your code with solutions to both problems. To hand in your code, type `cs141handin chess` from your working directory.

For each ILP formulation that changed from the ones you turned in for the integer-linear programming written assignment (or if you did not turn in the assignment), you must turn in an updated formulation. If a formulation did not change, you need not turn in an updated copy, but you must note this in your README. The formulations may be typed or hand-written, but must be legible. If it is typed you may include it with your code handin. Otherwise, state that you have a paper handin in your README and hand it into the handin bin on the second floor of the CIT.

Support Code

The first thing you want to do before starting this assignment is grab the support code from `/course/cs141/src/chess/` and place it in a directory called `chess`. You will find two `.java` files and a `build.xml` file. The `.java` files contain the classes in which you will implement your integer programs. They contain method stubs and sample interactions with the CPLEX libraries.

CPLEX is not freeware. After you have copied the support code, you must set an environment variable that tells CPLEX where it can find the necessary license information. This can be done by typing the following command into your working shell:

```
setenv ILOG_LICENSE_FILE /maytag/comm0/cplex/ilm/linux/access.ilm
```

Better, you may want to append this command to the `.environment` file in your home directory. Otherwise you will need to set this value in every shell in which you plan to use CPLEX.

You can compile your code at any time using the included build file, which can be invoked by typing the command `ant` into the terminal. You can run your code by typing the command `ant run`, and view the TA demo of the solution with `ant rundemo`. Since the program requires linking to external CPLEX libraries, just typing `javac *.java` won't properly compile it, so using the build file is highly recommended.

CPLEX

In order to use CPLEX for this assignment you will have to become familiar with the ILOG Concert Java interface. Most of the details you will need for this assignment are presented below; however if at any time you wish to explore other aspects of the Concert libraries you can visit

`file:///com/cplex/doc/html/index.html` within the CS department for API specifications (you'll mostly be working with the class `IloCplex`).

The `IloCplex` class is the main interface from your Java code to the CPLEX solver. The support code files instantiate an instance of this class at the beginning of each of the methods you must write. Your task is to formulate a model of your problem inside of the CPLEX solver represented by that `IloCplex` instance, so that it can be solved upon command. To do this you should call the methods on the `IloCplex` object described below. The `IloCplex` instance also acts as a *factory*: you will use it to instantiate instances of any other Concert classes you need for your program, rather than creating them yourself with the `new` command. Most of the modifications you make to the problem model via method calls on the `IloCplex` instance will also return instances of Concert classes, which you can store in instance variables for debugging purposes.

One warning: our version of CPLEX isn't fully Java1.5 compatible, so you should probably avoid using generics and other such fancy features. Sorry.

The main methods you will need to call on the `IloCplex` instance to shape your problem model inside the CPLEX solver and solve it are:

- `IloNumVar intVar(int lowerBound, int upperBound)`:
This method *only* returns an instance of an `IloNumVar`. It does not affect the problem model in any way. After calling this method, you'll want to store the returned instance in a variable so that you can access it later—this is the only way you can introduce new integer variables into your integer program. There are several different flavors of this method as explained in the API spec. The most useful of these variations is the `boolVar()` method, equivalent to the `intVar(int lowerBound, int upperBound)` method with a lower bound of 0 and an upper bound of 1. You won't need the variations that do not restrict the variables to integers.
- `IloLinearNumExpr linearNumExpr()`:
This method *only* returns an instance of `IloLinearNumExpr`. It does not affect the problem model in any way. `IloLinearNumExpr` instances are very useful tools for creating mathematical expressions of the type $ax_1 + bx_2 + \dots + c$ which can be added to your model with other methods. After calling this method, you'll want to store the resulting instance in a variable and then continually call the `IloLinearNumExpr.addTerm(IloNumVar x, double coef)` to add variables to the summation. In addition, you can use the `IloLinearNumExpr.setConstant(double c)` method to set the constant that is added in at the end of the expression. Using these linear expressions is not necessary; however, together with loops, they are useful for adding up multiple terms.
- `IloNumExpr sum(IloNumExpr a, IloNumExpr b)`:
This method returns an `IloNumExpr` instance that represents the sum of the two expressions passed in as arguments, again without affecting the problem model. There are several flavors of this method described in the API. The most useful variation is `IloNumExpr sum(IloNumExpr[] array)` which takes an array of `IloNumExpr` instances and returns an instance that represents the sum of all the elements of the array.

- `IloObjective addMaximize(IloNumExpr obj)`: This is the first method we've listed that actually modifies the internal model of the problem. It takes as its parameter an `IloNumExpr` which is a super class of almost all the Concert classes, including `IloNumVar` and `IloLinearNumExpr`. It then sets the *objective function* of the CPLEX model to be a maximization of the expression passed in as a parameter (`addMinimize(IloNumExpr obj)` does the same for minimization). It returns an instance of `IloObjective`, which represents the objective function and holds potential debugging information.
- `IloRange addLe(IloNumExpr lhs, double rhs)`:
This method adds constraints to your problem model. While this particular method call adds a "less than or equal to" constraint of the form $lhs \leq rhs$, there are several other versions of this method. `addGe(IloNumExpr lhs, double rhs)` adds a $lhs \geq rhs$ constraint, and `addEq(IloNumExpr lhs, double rhs)` adds a $lhs = rhs$ constraint. Make sure you don't leave off the word `add` in the method call or CPLEX will return an `IloRange` instance without adding it to the internal model. Any of these methods can be called with `double` arguments on either side, or `IloNumExpr` arguments on either side.
- `boolean solve()`:
This method invokes the CPLEX solver on your problem model. It will give control of the program flow to CPLEX until it finds a solution that it is happy with or a pre-determined time limit is reached. The boolean value returned indicates whether or not a *feasible* solution was found.
- `void end()`:
This method releases the `IloCplex` license held by the invoking object, and all the memory allocated by it. You **MUST** do this or you will lose points.
- `double getValue(IloNumVar v)`:
This method requests the value of a model variable from the CPLEX solver after a solution has been found. If a feasible solution has not been found, or the `solve()` method has not been called before this request is made, CPLEX will throw an exception. This is not the only way to request information about the solution (others are described in the API), but it should be sufficient for this assignment.

Extra Credit

Write an integer program to solve the *Knight's Tour Problem*, a variant of the traveling salesperson problem on a chess board: *can a knight visit all the squares on a chess board exactly once and return to its original position?*

Warning

CPLEX is a very expensive third-party software package. There are about 35 licenses available in the CS department for using CPLEX at any given time. Furthermore, other classes and researchers use these licenses, so if you wait until the last minute to work on this project you may find that you have to wait for others who are running software as well. The best advice we can give is to start early to minimize the possibility of this happening.

Resources

1. U.S. Chess Online
<http://www.uschess.org/beginners/letsplay.php>
2. LP FAQ
<http://www-unix.mcs.anl.gov/otc/Guide/faq/linear-programming-faq.html>