Cut Generation, Heuristics and Callbacks

How to Solve Hard IPs

Ross Anderson

Massachusetts Institute of Technology

Operations Research Center

15.S60 Software Tools for Operations Research

January 24, 2013

Overview

Advanced methods for integer programming:

- Generate constraints “on the fly”
- Suggest integer solutions

Case study: the Traveling Salesmen Problem (TSP)

- IP formulation with exponentially many constraints
- Heuristics well studied
  - Christofides
  - Two-Opt

Implementation

- CPLEX
- Java

How to take this tutorial

- Follow the course wiki
- These slides will help you get started
- Then follow me (I'll go slow), or use the wiki at your own pace
- New to Java? Ask your classmates (volunteers?)
- Really confused? Focus on concepts, not Java

A crash course in Java

Review topics:

- "Hello World!"
- Types
- Static methods
- Classes and files
- Objects
- Subclasses and inheritance
- Inner Classes
- Generics
- Primitives, Generics & Autoboxing
- Data Structures

Use this as a reference!

“Hello World!”

The simplest possible program!

Example.java

```java
public class Example {
  public static void main(String[] args) {
    System.out.println("Hello World!");
  }
}
```

“Hello World!”

The simplest possible program!

Example.java

```java
public class Example {
  public static void main(String[] args) {
    System.out.println("Hello World!");
  }
}
```

- All code must be in a class, we used `Example`
- Entry point is always `public static void main(String[] args)`

Variables have types

TypesExample.java

```
public class TypesExample {
  public static void main(String[] args){
    int x = 3;
    int y = 5;
    int z = x*y;
    System.out.println(z);//prints 15
    String s = "Hello ";
    String t = "World";
    s = s + t;
    System.out.println(s);//prints Hello World
    //Type safety, won't compile
    //String makesNoSense = x*s;
  }
}
```

Static methods

StaticMethods.java

```java
public class StaticMethods {
  public static void main(String[] args){
    int z = StaticMethods.sum(3,4);
    System.out.println(z);//prints 7
  }
  public static int sum(int x, int y){
    return x + y;
  }
}
```

- Argument types, return type
- Invoke with *[class name].[method name]([arguments])*

Every class has a file

Calculator.java

```
public class Calculator {
  public static double product(double x, double y){
    return x*y;
  }
}
```

Calculations.java

```
public class Calculations {
  public static void main(String[] args) {
    double x = Calculator.product(2, 3.5);
    System.out.println(x);//prints 7
  }
}
```

- Entry point still `public static void main(String[] args)`
- Stay organized!

Objects

Person.java

```java
public class Person {
  //fields
  private int age;
  private String name;
  //constructor
  public Person(int age, String name) {
    super();
    this.age = age;
    this.name = name;
  }
  //non-static methods
  public int getAge() {
    return age;
  }
  public String getName() {
    return name;
  }
  public void increaseAge(){
    age = age+1;
  }
}
```

ObjectExample.java

```java
public class ObjectExample {
  public static void main(String[] args){
    Person ross = new Person(25,"Ross");
    //prints 25
    System.out.println(ross.getAge());
    //prints 26
    ross.increaseAge();
    System.out.println(ross.getAge());
  }
}
```

- *Constructors* create *instances* of objects
- Call *methods* for an instance with *[instance].[method]([arguments])*

Subclasses and Inheritance

American.java

```java
public class American extends Person{
  private int socialSecurity;
  public American(int age, String name, int socialSecurity) {
    //must begin with superclass constructor
    super(age, name);
    this.socialSecurity = socialSecurity;
  }
  public int getSocialSecurity(){
    return socialSecurity;
  }
}
```

SubclassExample.java

```java
public class SubclassExample {
  //all ages are approximate
  public static void main(String[] args) {
    American ross = new American(25,"ross",123);
    //increaseAge() is "inherited" from Person
    ross.increaseAge();
    Person vishal = new American(40,"vishal",321);
    //does not compile
    //vishal.getSocialSecurity();
    //American iain = new Person(23,"iain");
  }
}
```

- Every `American` is a `Person`
- Not every `Person` is an `American`

Inner Classes

Company.java

```
public class Company {
  private String name;
  public Company(String name){
    this.name = name;
  }
  public class CompanyLocation{
    private String location;
    public CompanyLocation(String location){
      this.location = location;
    }
    public String getDescription(){
      return name + " at " + location;
    }
  }
  public CompanyLocation makeLocation(String location){
    return this.new CompanyLocation(location);
  }
}
```

- Every instance of inner class associated with some instance of outer class
- Inner class can accesses fields and methods of associated outer class instance

Generics

Wrapper.java

```
public class Wrapper<T> {
  private T value;
  public Wrapper(T value){
    this.value = value;
  }
  public T getValue(){
    return value;
  }
  public void setValue(T value){
    this.value = value;
  }
}
```

- The class Wrapper is parametrized by another class.
- Reuse code
- Type safe!

GenericsExample.java

```
public class GenericsExample {
  public static void main(String[] args){
    Wrapper<String> wrapper = new Wrapper<String>("hello");
    String s = wrapper.getValue();
    //does not compile
    //wrapper.setValue(7);
    //int x = wrapper.getValue()
  }
}
```

Primitives, Generics & Autoboxing

```
//does not compile
//Wrapper<int> intWrapper = new Wrapper<int>(7);
Wrapper<Integer> integerWrapper = new Wrapper<Integer>(new Integer(7));
int val = integerWrapper.getValue().intValue();
//autoboxing!  type conversion automatic
Wrapper<Integer> integerWrapper2 = new Wrapper<Integer>(7);
//auto-un-boxing! type conversion automatic
int val2 = integerWrapper2.getValue();
```

- Primitives take less memory
- Overhead of `Integer` usually irrelevant

Primitive Type	Class
`int`	`Integer`
`double`	`Double`
`boolean`	`Boolean`

Data Structures

Sample data structure usage

```
String[] array = new String[3];//fixed size
array[0] = "first";
array[2] = "last";
//array[3]  causes exception
//array[1].length() causes exception
List<String> list = new ArrayList<String>();
list.add("hello");//size changes automatically
list.get(0);//evaluates to hello
Set<String> set = new HashSet<String>();
set.add("hello");
set.contains("hello");//evaluates to true
for(String s: set){
  System.out.println(s);//prints hello
}
Map<String,Integer> map = new HashMap<String,Integer>();
map.put("hello", 1);
map.put("world", 2);
map.get("world");//evaluates to 2
Set<String> keys = map.keySet();//the set {hello,world}
```

- Arrays are fast but not flexible
- Use Lists, Sets, and Maps, when possible
- *Interface* specifies behavior
- Class *implements* interface
- “Program to the interface!”

Interface	Implementation
List	ArrayList
Set	HashSet
Map	HashMap

`Warmups.exerciseOne()`, learn by example!

$$\begin{array}{ll} \text{min} & a \\ \text{subject to} & 5a \geq 2 \\ & a \in \{0,1\} \end{array}$$

Partial Solution Snippet

```
IloCplex cplex = new IloCplex();
IloIntVar a = cplex.boolVar();
cplex.addMinimize(a);//minimize a
IloLinearIntExpr sum = cplex.linearIntExpr();
sum.addTerm(a, 5);//5*a
cplex.addGe(sum, 2);//subject to 5*a >= 2
```

Solve `exerciseOne`! Use the wiki documentation!

Numerical tolerance for `cplex.getValue()`

Take care when checking integer variables!

Bad

```java
IloCplex cplex = new IloCplex();
IloIntVar var = cplex.boolVar();
//...
//solve some problem
//...
double val = cplex.getValue(var);
if(val == 0){
  //take some action
}
```

Good

```java
double val = cplex.getValue(var);
if(Math.abs(val) < 0.00001){
  //take some action
}
//but we do this for you
//see tspSolver/src/Util.java
if(Util.doubleToBoolean(val)){
  //take some action
}
```

Warning: cplex.getValue()

CPLEX is designed for bulk actions on arrays:

Array Speed

```
IloIntVar[] variables = cplex.boolVarArray(2000000);
//fast
double[] vals = cplex.getValues(variables);
//over 10 times slower!
double[] vals2 = new double[2000000];
for(int i = 0; i < variables.length; i++){
  vals2[i] = cplex.getValue(variables[i]);
}
```

- Usually moot, IP solving time dominates
- Java style, avoid arrays

Avoid indices

Common problem: one variable per X

1. Lists/arrays allow indexing error
2. Multiple data structures out of sync
3. Use a single Bi-Directional Map!

Bad– Indices

```
List<String> strings = makeStrings();
IloIntVar[] vars = cplex.boolVarArray(strings.size());
List<String> constraintStrings = relevantSubset(strings);
IloLinearIntExpr sum = cplex.linearIntExpr();
for(String s : constraintStrings){
  sum.addTerm(vars[strings.indexOf(s)], 1);
}
```

Good– BiMap

```
List<String> strings = makeStrings();
IloIntVar[] vars = cplex.boolVarArray(strings.size());
BiMap<String,IloIntVar> map = HashBiMap.create();
for(int i = 0; i < strings.size(); i++){
  map.put(strings.get(i), vars[i]);
}
List<String> constraintStrings = relevantSubset(strings);
IloLinearIntExpr sum = cplex.linearIntExpr();
for(String s : constraintStrings){
  sum.addTerm(map.get(s), 1);
}
```

Use Util

```
BiMap<String,IloIntVar> map = Util.makeBinaryVariables(cplex,strings);
```

`Warmups.exerciseTwo()`, Java Style

Recode the IP from `exerciseOne()`

$$\begin{aligned} \text{min} \quad & x + 2y + 3z \\ \text{subject to} \quad & x + y + z \geq 2 \\ & x, y, z \in \{0, 1\} \end{aligned}$$

using `Util.makeBinaryVariables()` and both versions of `Util.integerSum()`

Do it yourself!

- You know everything you need to know! Follow the wiki
- I will work very slowly
- We will stop to talk and discuss
- Go ahead or help your neighbor if you work faster than me