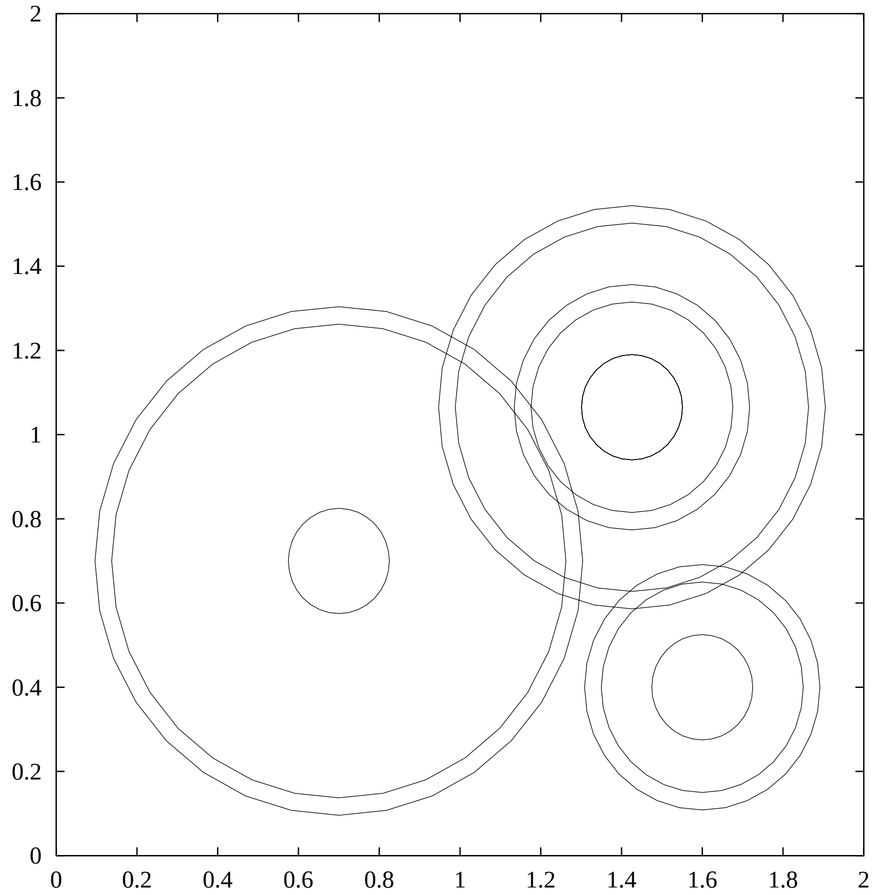
CS533 — Written Homework 2 — Solution

1. [45] Consider the following gear-box design problem. The input-to-output ratio should be 63/16. The dimensions of the box are 2.2 inches square with the lower left corner at the origin. The input shaft should be centered at (0.8, 0.8) and the output shaft should be centered at (1.7, 0.4). Assume the radius of each shaft is 0.125 and that each shaft passes completely through the gear box. Use the gears listed on the slide distributed in class. Hint: I believe a OneTwoOne design is required.

We can use four gears.
The input gear will have 27 teeth. It will mesh with Idler1Gear1 having 12 teeth. On that same shaft will be Idler1Gear2 with 21 teeth. It will mesh with outGear having 12 teeth. (27/12 * 21/12) = 63/16.

The idler shaft must be located at (1.56487, 1.07409). Here is a plot of the solution:



2. [10] Write a rule that would test for a collision between a gear and a shaft. Note, if you need to use two parameters of the same type in the condition, you can use the syntax NEEDS(type, name1) and NEEDS(type,name2) to give them two different names (name1 and name2). Assume that the radius of each shaft is 0.125 inches.

BEGINTEST(Rule_GearHitsShaft)
NEEDS(Gear) NEEDS(Shaft, Shaft1) NEEDS(Shaft, Shaft2)

```
IF( (Gear == KNOWN) &&
   (Shaft1 == KNOWN) &&
   (Shaft2 == KNOWN) &&
   (! (Shaft1 == Shaft2)) &&
   MountedOn(Gear, Shaft1)
)
THEN(
  test(Gear().OuterRadius + 0.125 ≥ mindistance(Shaft1(),Shaft2()),
      "Gear collides with other shaft");
   )
ENDRULE
```

3. Consider the following design problem (modified from Gonzalez & Dankel, 1995): Suppose that John Smith, a student, is changing apartments and needs to move his belongings (listed in the table below). He must pack them into boxes subject to the following constraints.

- The total number of boxes should be minimized (at least approximately).
- All fragile items must be padded.
- If a box contains a fragile item, then it cannot contain any objects heavier than that item.
- The sum of the sizes of the objects in the box cannot exceed 10 units.
- Every item must be packed in some box.

Item	Count	Weight	Size	Fragile?
TV	1	heavy	10	yes
Books	7	heavy	2	no
VCR	1	medium	4	no
Lamp	2	medium	8	yes
Set of Dishes	1	medium	5	yes
Set of Glasses	1	medium	4	yes
coat	2	light	3	no
Jeans	4	light	2	no
shirts	8	light	1	no

(a) [5] Analyze each of the constraints in terms of their degree of locality.

- The total number of boxes should be minimized (at least approximately). This is not really a constraint; it is the objective function.
- All fragile items must be padded. This is a very local constraint. It can be applied to each item independent of other items.
- If a box contains a fragile item, then it cannot contain any objects heavier than that item. This is an intermediate constraint. It can be applied to any partially-filled box.
- The sum of the sizes of the objects in the box cannot exceed 10 units.
  This is an intermediate constraint. It can only be applied to a full box.

- Every item must be packed in some box. This is a global constraint, but it can be checked on an item-by-item basis.

(b) [10] Assume that a state is represented by two items: (i) list of boxes where box is a list of objects and each object can be padded or unpadded and (ii) a list of objects remaining to be boxed. Write C++ class definitions to represent such states. Suggestion: represent each object separately (i.e., there are 7 books, so there should be 7 distinct book objects).

```
enum Weight {light, medium, heavy};

class object
{
public:
  string name;
  Weight weight;
  int   size;
  int   isFragile;
  int   isPadded;
  Box * inBox;
};

class box
{
public:
  list<object> contents;
};

class state
{
public:
  list<object> unpacked;
  list<box> boxes;
};
```

(c) [10] Write a C++ function to determine whether a state is a feasible solution.

```
int box::totalsize()
{
  // returns the total size of the objects in the box
  listIterator<object> itr(contents);
  int result = 0;
  for(itr.init(); !itr; ++itr) {
    result += itr().size;
  }
  return result;
}

int box::mixesFragileAndHeavy()
{
```

```
  int containsFragile = 0;
  int containsMediumOrHeavy = 0;
  listIterator<object> itr(contents);
  for(itr.init(); !itr; ++itr) {
    if (itr().isFragile) containsFragile = 1;
    if ((itr().weight == medium) ||
        (itr().weight == heavy)) containsMediumOrHeavy = 1;
  }
  return (containsFragile && containsMediumOrHeavy);
}

int state::feasible()
{
  if (!unpacked.isEmpty()) return 0;
  listIterator<box> boxitr(boxes);
  for(boxitr.init(); !boxitr; ++boxitr) {
    if(boxitr().totalsize() > 10) return 0;
    if(boxitr().mixesFragileAndHeavy()) return 0;
  }
  return 1;
}
```

(d) [10] Describe a constructive problem space for this problem. Describe your operators in pseudocode. The problem space should not do any constraint propagation.

The starting state would have all objects on the unboxed list and nothing on the boxes list.

There would be two operators.

- Add new box.
- Choose an unboxed object and put it into an existing box.

(e) [10] For each of the constraints given above, indicate whether it can be used to propagate information. If so, write a pseudocode description of a rule that would propagate this information.

- The total number of boxes should be minimized (at least approximately).
  This is the objective function. It can be approximately minimized by a greedy algorithm that always prefers to place unboxed objects in an existing box rather than starting a new box.
- All fragile items must be padded.
  This can be immediately propagated by padding all of the fragile items.
- If a box contains a fragile item, then it cannot contain any objects heavier than that item. This can be propagated by forcing all new fragile items to be placed in boxes that do not contain any objects heavier than them. Also, non-fragile objects must not be placed in boxes that already contain fragile objects lighter than them. Basically, propagation here just involves restricting the set of potential boxes to consider for an item.

  IF unboxed item is fragile and there exists a box containing fragile objects of the same weight
  THEN place the item in that box.

  IF unboxed item is fragile and there exists a box containing no fragile objects but with all objects having the same or lighter weight
  THEN place the item in that box.

IF unboxed item is not fragile and there exists a box containing fragile objects of the same or greater weight
THEN place the item in that box.

IF unboxed item is not fragile and there exists a box that does not contain any fragile objects
THEN place the item in that box.

- The sum of the sizes of the objects in the box cannot exceed 10 units.
  This can be propagated by keeping track of the remaining space in the box and not adding items that would drive the total above 10 units.

  We need to incorporate this into the rules from the previous constraint. Each of the previous rules should have the additional condition that the box has room for the item. Each of the rules should also have the additional action that the amount of available space in the box should be decremented by the weight of the new item.

- Every item must be packed in some box.
  This can't be propagated, but it can be easily incorporated by having a rule that matches any unboxed item.