Graduate Artificial Intelligence 15-780

Homework #2: *SOLUTIONS*

Out on February 13
Due on February 27

Problem 1: Planning [60 pts]

I. STRIPS

Using the basic grounded STRIPS, write a planning domain to solve a specific planning task of your choice.

a) The domain should have at least 4 types of operators.

b) Show two versions of the domain with different sets of operators.

c) Run a planner of your choice in these two different versions. Wesley Tansey from the University of Texas at Austin has posted an easy-to-use STRIPS planner (written in Python) here: https://github.com/tansey/strips.

Please submit a printout of both versions of your domain, printouts of two problems in your domain (i.e., the initial state and goal state that define each problem), and printouts of the output of your choice planner for both these problems each using both versions of the domain. Discuss the main choices you made in representing the state space and the operators, and discuss how the different versions of the domain affect the efficiency of solving problems. **20 pts**

—SOLUTIONS—

There's no "right" answer to this problem, since we're asking for creativity! However, there are three sample domains shown in this solution. I used an old planner (written when I took Manuela's class!) to plan, but your results should look similar.

Below is a simple translation of the well-known Berkeley sockets API to a STRIPS domain. A client creates a socket, connects to a server, and sends or receives data. A server creates a socket, binds that socket to an address, listens on the socket for incoming connections, then blocks on a connection while sending and receiving data. The server repeats this listening process *ad infinitum* for many clients.

This section presents two versions of the same domain. Each uses the same base literals but presents different sets of operators. Both versions of the domain are used to solve both problems explained below. It also briefly presents a simple three-block BlocksWorld instantiation, used to optimally solve Sussman's anomaly, as well as a solution to the One Way Rocket problem.

The first example problem in my novel domain is that of turning on a mobile client, connecting to a server that is already listening, and getting some data. The second example problem expands to a general situation with more than one client getting information, along with some initialization for the server.

Original Domain: Berkeley Sockets

Conditions

CLIENT1_IDLE = "Client #1 Computer On";
CLIENT2_IDLE = "Client #2 Computer On";
SERVER_IDLE = "Server Computer On";
FREE_SOCKET_CLIENT1 = "Client #1 has free socket";
FREE_SOCKET_CLIENT2 = "Client #2 has free socket";
FREE_SOCKET_SERVER = "Server has free socket";
BOUND_SOCKET_SERVER = "Server has bound socket to address";
SERVER_LISTENING = "Server is listening on bound socket";
CLIENT1_CONNECTED = "Client #1 is connected to server";
CLIENT2_CONNECTED = "Client #2 is connected to server";
SERVER_BLOCKING = "Server is blocking";
REQUEST_SENT_CLIENT1 = "Client #1 sent request";

REQUEST_SENT_CLIENT2 = “Client #2 sent request”;
RESPONSE_SENT_TO_CLIENT1 = “Server sent response to Client #1”;
RESPONSE_SENT_TO_CLIENT2 = “Server sent response to Client #2”;

Example Problem 1

Initial State	Goal State
CLIENT1_IDLE SERVER_LISTENING	RESPONSE_SENT_TO_CLIENT1

Example Problem 2

Initial State	Goal State
CLIENT1_IDLE CLIENT2_IDLE SERVER_IDLE	RESPONSE_SENT_TO_CLIENT1 RESPONSE_SENT_TO_CLIENT2

Operator Set 1

```
OP_CREATE_SOCKET_CLIENT1
Pre:  CLIENT1_IDLE
Del:  CLIENT1_IDLE
Add:  FREE_SOCKET_CLIENT1

OP_CREATE_SOCKET_CLIENT2
Pre:  CLIENT2_IDLE
Del:  CLIENT2_IDLE
Add:  FREE_SOCKET_CLIENT2

OP_CREATE_SOCKET_SERVER
Pre:  SERVER_IDLE
Del:  SERVER_IDLE
Add:  FREE_SOCKET_SERVER

OP_BIND_SOCKET_SERVER
Pre:  FREE_SOCKET_SERVER
Del:  FREE_SOCKET_SERVER
Add:  BOUND_SOCKET_SERVER

OP_LISTEN_SOCKET_SERVER
Pre:  BOUND_SOCKET_SERVER
Del:  BOUND_SOCKET_SERVER
Add:  SERVER_LISTENING

OP_CONNECT_CLIENT1
Pre:  SERVER_LISTENING
      FREE_SOCKET_CLIENT1
Del:  SERVER_LISTENING
      FREE_SOCKET_CLIENT1
Add:  SERVER_BLOCKING
      CLIENT1_CONNECTED

OP_CONNECT_CLIENT2
Pre:  SERVER_LISTENING
      FREE_SOCKET_CLIENT2
Del:  SERVER_LISTENING
      FREE_SOCKET_CLIENT2
Add:  SERVER_BLOCKING
      CLIENT2_CONNECTED

OP_SEND_REQUEST_CLIENT1
Pre:  CLIENT1_CONNECTED
Del:
Add:  REQUEST_SENT_CLIENT1

OP_SEND_REQUEST_CLIENT2
Pre:  CLIENT2_CONNECTED
Del:
Add:  REQUEST_SENT_CLIENT2
```

`OP_SEND_RESPONSE_TO_CLIENT1`
Pre: `REQUEST_SENT_CLIENT1`
Del: `REQUEST_SENT_CLIENT1`
Add: `RESPONSE_SENT_TO_CLIENT1`

`OP_SEND_RESPONSE_TO_CLIENT2`
Pre: `REQUEST_SENT_CLIENT2`
Del: `REQUEST_SENT_CLIENT2`
Add: `RESPONSE_SENT_TO_CLIENT2`

`OP_CLOSE_CONNECTION_TO_CLIENT1`
Pre: `CLIENT1_CONNECTED` `SERVER_BLOCKING`
Del: `CLIENT1_CONNECTED` `SERVER_BLOCKING`
Add: `SERVER_LISTENING` `CLIENT1_IDLE`

`OP_CLOSE_CONNECTION_TO_CLIENT2`
Pre: `CLIENT2_CONNECTED` `SERVER_BLOCKING`
Del: `CLIENT2_CONNECTED` `SERVER_BLOCKING`
Add: `SERVER_LISTENING` `CLIENT2_IDLE`

Solving Problems in Operator Set 1

First, we solve Problem #1. This problem gets a single client to grab some data from a ready-and-waiting server.

```
OP: Create Socket Client #1
OP: Client #1 attempting to connect
OP: Client #1 sending request to Server
OP: Server sending response to Client #1
```

Next, we solve the (harder) Problem #2. This problem gets more than one client to chat with a server. The server is also forced to initialize itself; it is not "ready-and-waiting" from the start.

```
OP: Create Socket Client #2
OP: Create Socket Client #1
OP: Create Socket Server
OP: Bind Socket Server
OP: Listen Socket Server
OP: Client #2 attempting to connect
OP: Client #2 sending request to Server
OP: Server sending response to Client #2
OP: Server is resetting connection to Client #2
OP: Client #1 attempting to connect
OP: Client #1 sending request to Server
OP: Server sending response to Client #1
```

As a human looking at the plan, one can see some trivial "partial-ordering" that could occur in this domain. For instance, it does not matter in what order the clients and server create their sockets. However, the sockets must be created before any initialization or networking occurs over them. Similarly, it does not matter if Client #1 talks with the Server before Client #2 or vice versa.

Operator Set 2

`OP_CREATE_SOCKET_ALL_CLIENTS`
Pre: `CLIENT1_IDLE` `CLIENT2_IDLE`
Del: `CLIENT1_IDLE` `CLIENT2_IDLE`
Add: `FREE_SOCKET_CLIENT1` `FREE_SOCKET_CLIENT2`

`OP_CREATE_SOCKET_SERVER`
Pre: `SERVER_IDLE`
Del: `SERVER_IDLE`
Add: `FREE_SOCKET_SERVER`

`OP_READY_SOCKET_SERVER`
Pre: `FREE_SOCKET_SERVER`
Del: `FREE_SOCKET_SERVER`
Add: `SERVER_LISTENING`

`OP_GET_DATA_CLIENT1`
Pre: `SERVER_LISTENING`
`FREE_SOCKET_CLIENT1`
Del: `FREE_SOCKET_CLIENT1`
Add: `RESPONSE_SENT_TO_CLIENT1`
`CLIENT1_IDLE`

`OP_GET_DATA_CLIENT2`
Pre: `SERVER_LISTENING`
`FREE_SOCKET_CLIENT2`
Del: `FREE_SOCKET_CLIENT2`
Add: `RESPONSE_SENT_TO_CLIENT2`
`CLIENT2_IDLE`

Solving Problems in Operator Set 2

Again, we solve Problem #1. This problem gets a single client to grab some data from a ready-and-waiting server. Interestingly, with this new set of operators, Problem #1 becomes unsolvable. Since we do originally did not specify an initial state for Client #2, we cannot successfully call the `OP_CREATE_SOCKET_ALL_CLIENTS`. This type of issue is discussed in the Discussion section. For now, we add a meaningless `CLIENT2_IDLE` to the initial state.

```
OP: Create Sockets for all Clients
OP: Retrieving data for Client #1
```

Next, we solve the (harder) Problem #2. This problem gets more than one client to chat with a server. The server is also forced to initialize itself at the start. `OP: Create Sockets for all Clients`

```
OP: Create Socket Server
OP: Setting up Socket Server
OP: Retrieving data for Client #2
OP: Retrieving data for Client #1
```

Discussion

I chose the set of literals used to represent the state space, perhaps naïvely, by looking at a flow chart of the Berkeley sockets API. With a few exceptions, it is fairly easy to discern at what "state" a Client or Server is based on where in the flow chart it is located.

This relatively high-level state representation would fall apart were more realistic assumptions to be introduced. For instance, this representation doesn't deal with the simple race conditions that occur in networking. For instance, if two Clients send connection requests to a Server simultaneously (*i.e.*, they both see `SERVER_LISTENING`), this representation will have them both believe they are connected, while the Server will have rejected one of them.

# Nodes	Problem 1	Problem 2
Operator Set 1	5	156445
Operators Set 2	3	96

Timing (s)	Problem 1	Problem 2
Operator Set 1	0.000	1.529
Operators Set 2	0.000	0.005

Figure 1: Node expansion and timing results for both versions of the Berkeley Sockets domain

Conversely, being *less* descriptive with the state and operator space description – for instance, with Operator Set 2's formation of one action *A* out of the {Connect, Request, Respond, Disconnect} cycle – removes much problem solving difficulty from this domain. After socket initialization, multiple clients can perform their respective actions *A* in any order whatsoever, indefinitely. However, as we saw with the first example problem, abstracting the operations *too* much (without abstracting the state space appropriately) can provide some unintuitive and wasteful initial and final goal specifications.

The simplicity of the domain representation has a direct effect on problem solving efficiency (see Figure 1). In the "too easy" representation above, solving a problem is (unrealistically) simple and fast, since nearly an operation is legal at any time. However, in the "too realistic" representation, problem solving efficiency would be impacted incredibly, especially with the introduction of latency and time. My poor planner would be forced to discretize time (perhaps to the millisecond scale). With timeouts of 120 *seconds* being common in the networking world, the state and operator space would explode into intractability. Overall, I feel this state space is a happy medium between "too easy" and "too realistic (and thus too complicated)."

Other Domains: Blocks World

Below is some output of my planner's solutions to an easy problem in Blocks World, as well as an optimal solution to Sussman's Anomaly.

```
***========================================***

InstanceNonlinear_BLOCKSWORLD_Normal
Initial state:  [C_on_A, A_on_Table, Clear_C, B_on_Table, Clear_B, Hand_Empty]
Goal state:  [C_on_B, B_on_Table, A_on_C]

Experiment ran 10 times.
Avg.  Nodes expanded:  543
Avg.  time spent (wall, s):  0.017

Sample plan found:
Pickup_C_from_A
Place_C_on_B
Pickup_A_from_Table
Place_A_on_C

***========================================***

InstanceNonlinear_BLOCKSWORLD_Sussmans_Anomaly
Initial state:  [C_on_A, A_on_Table, Clear_C, B_on_Table, Clear_B, Hand_Empty]
Goal state:  [C_on_Table, B_on_C, A_on_B]

Experiment ran 10 times.
Avg.  Nodes expanded:  21281
Avg.  time spent (wall, s):  0.225
```

```
Sample plan found:
Pickup_C_from_A
Place_C_on_Table
Pickup_B_from_Table
Place_B_on_C
Pickup_A_from_Table
Place_A_on_B
```

Other Domains: One Way Rocket

Below is some output of my planner's solutions to the One Way Rocket domain [3]. Due to its inability to interleave goals, a linear planner fails to find a solution for this problem; however, a nonlinear planner has no such issue.

```
***==========================================***

InstanceNonlinear_ONE_WAY_ROCKET
Initial state:  [Object 2 at Location A, Rocket at Location A, Rocket has fuel, Object 1 at
Location A]
Goal state:  [Object 2 at Location B, Object 1 at Location B]

Experiment ran 10 times.
Avg.  Nodes expanded:  193
Avg.  time spent (wall, s):  0.009

Sample plan found:
Loading Object 1 into Rocket at Location A
Loading Object 2 into Rocket at Location A
Moving Rocket from Location A to Location B
Unloading Object 2 from Rocket at Location B
Unloading Object 1 from Rocket at Location B
```

II. Partial-order planning

Partial-order planning backtracks over which action to use to achieve an open condition and which method to use to resolve a threat (flaw). It does not, however, backtrack over the order in which to achieve open conditions, nor the order in which to resolve threats. For each of the questions below, please explain your answers precisely and thoroughly.

—SOLUTIONS—

a) Why doesn't the fact that open conditions can be achieved in any order affect the comple teness of partial-order planning? **5 pts**

Partial-order planning treats open conditions as commutative conditions; the order in which they are addressed will not affect correctness. This is often an advantage over linear planners, where the combinatorial possible orderings of "parallelizable" operators can cause an exponential number of tree searches.

The reason why this commutative assumption does not affect completeness is that partial-order planners, unlike linear planners, only impose orderings/constraints when *absolutely necessary*. Linear planners, trapped by their forced ordering of goals, often constrain a problem more than it should; for instance, the One Way Rocket [3] problem is easily solved by partial-order planners, but is not solvable by linear planners. In short, if a POP decides not to expand a specific ordering of conditions, that ordering of conditions is *illegal* – thus, the planner's completeness remains.

b) Why doesn't the fact that threats can be resolved in any order affect the completeness of partial-order planning? **5 pts**

Although partial-order planners backtrack over *how* to resolve a specific threat, the same commutative property described above applies to the *order* in which to resolve threats. If a partial-order planner adds an ordering constraint to a set of threats, it is because other orderings would never appear in a solution anyways.

c) Give, and explain, an example of how the order in which open conditions are achieved affects the efficiency of partial-order planning. **5 pts**

For our example, let there be only two open conditions. Also, assume we already have a partial plan p; that is we've already chosen some operators to achieve some other goals. Assume one of our open conditions is infeasible given the partial plan p. Assume the other open condition is feasible given p. Furthermore, let the infeasibility of the first condition be easy to determine, while the feasibility of the second condition is difficult.

Then, if our partial-order planner decides to solve the second condition before the first, it will prove feasibility after a lot of work before quickly determining the infeasibility of the current plan p based on the first open condition. Conversely, if the planner decides to chase down the first condition before the second, it will quickly realize that p is infeasible, and it will backtrack.

d) Give, and explain, an example of how the order in which threats are resolved affects the efficiency of partial-order planning. **5 pts**

For this answer, see the example above. Adjust the first open condition – the condition whose infeasibility was "easy" to determine – as follows: if some threat t_A is tackled first, infeasibility is "easy" to prove; however, if some other threat t_B is tacked first, infeasibility is not obvious.

Then, if the partial-order planner decides to resolve the threat t_B before t_A, it will spend a ton of time resolving t_B only to quickly fail to resolve t_A, leading to backtracking. Conversely, if the planner attempts to resolve t_A first, it will quickly prove infeasibility and backtrack.

op_name:	o_1^p	o_2^p	o_3^p	o_4^p	o_5^p
pre:				y	x
del:		x	y		
add:	c_1	c_2	c_3	c_4	$c_3 \wedge c_4$

Figure 2: 3-wise exclusive operators o_3^p, o_4^p, o_5^p prevent O^b from simultaneous execution.

III. GraphPlan

—SOLUTIONS—

a) GraphPlan can backtrack during its second phase while searching backwards from the goals through the planning graph. Explain, in words and with a simple example, why GraphPlan needs to backtrack in its backwards search. **10 pts**

In Section 2.2 of [1], Blum and Furst provide effective heuristics for discovering almost all *pairwise* mutual exclusion relationships between either operators (at the operator-level) or propositions (at the proposition-level). While determining all n-wise exclusion relationships would be fatal in terms of runtime, this ignorance of more complex exclusion forces GraphPlan to perform backtracking.

Let there be a set of goals $g = \{g_1, g_2, g_3\}$. Assume g is satisifed by two groups of three operators each: call them $O^a = \{o_1^a, o_2^a, o_3^a\}$ and $O^b = \{o_1^b, o_2^b, o_4^b\}$. Each operator in $o_i^a \in O^a$ and $o_i^b \in O^b$ requires a precondition c_i, adds a goal g_i, and then deletes that same c_i – except o_4^b, which adds g_3. So, when either all of O^a or O^b exists at the same level in the planning graph, GraphPlan will halt and backwards search for a plan.

To get this to happen, we need some operators that add preconditions c_1, c_2, c_3, and c_4. Here is where the pairwise exclusion limitation comes into play. Assume the following extra operators, *of which no pair is mutually exclusive*:

Assume that conditions x and y are true at the beginning. Then, we see that the set $\{c_2, c_3, c_4\}$ – which just happen to be the preconditions for set O^b – cannot be turned true at exactly the same time. However, GraphPlan does not recognize this immediately, since it only looks at exclusion *pairs*. Were it to look at exclusion *triples*, in this case it would catch the relationship.

Also, note that c_1, c_2, and c_3 can be achieved simultaneously. These are the preconditions to set O^a.

Then, at a time step t, GraphPlan sees that the goal set exists on the same level. Assume it decides to backward search in a way that leads to operator set O^b. Only now will it see that there is no way for required preconditions c_2, c_3, and c_4 to become true in the same timestep. GraphPlan will be forced to backtrack and, eventually, will lead to O^a and the preconditions c_1, c_2, c_3, which are satisfied in a single step via their respective operators.

b) If GraphPlan cannot find a solution during the backwards search, does this mean that the problem is not solvable? Carefully justify your answer. **5 pts**

From page 5 of Blum and Furst [1], "if a valid plan exists using t or fewer time steps, then that plan exists as a subgraph of the planning graph." This is a key aspect of graph plan's ability to, during backward searching at a time t, prove that no plan exists with fewer than t time steps. This says nothing of possible plans at times greater than t; in fact, the Graph Plan algorithm will keep on searching until it either (a) finds a path with greater than t time steps or (b) terminates after the planning graph "levels off" and no complete plan is found.[1]

c) Is it possible for GraphPlan to terminate after finding an n time step plan of a total of k operators, while there actually exists an n time step plan with less than k operators in the planning graph? If your answer is

[1] See Section 5.3 of Blum and Furst [1] for the termination proof.

positive, show a concrete example and explain the general case. If your answer is negative, explain why not?
5 pts

Yes. My example relies on the nondeterminism inherent in GraphPlan. Imagine a group of $k > 1$ operators o_i, each adding a goal g_i, each with no preconditions or deletes. Now imagine some operator `MEGA_OP`, also with no preconditions or deletes, that adds all goals $\{g_1, \ldots, g_k\}$. Then GraphPlan would terminate after $t = 1$ time steps, but could pick plan $\{o_1, \ldots, o_k\}$ over plan {`MEGA_OP`} – *i.e.*, it could pick the plan with $k > 1$ operators over the plan with exactly 1 operator.

Problem 2: Search & Integer Programming [40 pts]

In the United States, any individual wanting to broadcast using radio waves must first acquire an operating *license* for a certain frequency range in a certain geographic region. You were recently hired by the Federal Communications Commission (FCC), the regulatory body in charge of the spectrum, to conduct an auction to sell licenses to potential buyers.

The wireless spectrum is discretized into m disjoint segments of frequency range, $\{\ell_1, \ell_2, \ldots, \ell_m\}$. Each of the n potential buyers b express a *valuation* for combinations of these frequency ranges through `XOR` bids, an exclusive disjunction over conjunctions of spectrum segments. An example bid by bidder i is:

$$\langle (v_1^b, \ell_1 \wedge \ell_3) \oplus (v_2^b, \ell_6) \rangle$$

In this example, the bidder announces a valuation of v_1^b for ownership of both segments ℓ_1 and ℓ_3, and a valuation of v_2^b for ownership of segment ℓ_6. The $\oplus$ constraint between the spectrum conjunctions enforces the rule that only *one* of these conjunctins can be one by the bidder.

Solving the winner determination problem

The *winner determination problem* (WDP) is that of finding an allocation of items (in this case, spectrum licenses) to bidders that maximizes the auctioneer's revenue. The auctioneer's revenue is maximized by choosing an allocation that maximizes the sum (over all bidders) of the bidders' valuations for the subset of items they receive.

a) Formulate the WDP for the FCC spectrum auction as an integer linear program (IP). **5 pts**

—SOLUTIONS—

First, we define the decision variables for the problem. Second, we write the objective (revenue maximization) in the language of these variables. Third and finally, we describe sets of constraints that ensure the feasibility of the final solution.

Variables

Each wireless segment can be allocated to at most one bidder; however, bidders express their preferences over conjunctions of segments, not individual segments. Let the set of bidders be $B = \{1, \ldots, n\}$ and the set of wireless segments be $L = \{1, \ldots, m\}$. Then we can refer to a conjunction of segments as $S \subseteq L$. That is, if a bidder i bids on ℓ_1 and ℓ_3, and there are 3 segments available, then $S = \{1,3\} \subseteq \{1,2,3\} = L$. Finally, let $v_i(S)$ be the valuation of bidder i for the conjunction S; this is expressed in the bidder's bid as well.

We define integer (binary) variables $x_i(S) \in \{0,1\}$ for each conjunction S and each bidder i, where $x_i(S) = 1$ if bidder i receives conjunction S, and $x_i(S) = 0$ otherwise (the bid is not accepted). Note that this results in a potentially exponential number of decision variables (because we're defining, possibly, $B \cdot 2^L$ variables); however, in practice, bidders do not bid on *every*—or even close to every—conjunction, so we ignore those decision variables that don't match with an actual bid.

For a more in-depth look at formulating the WDP (in general) as an IP, check out [2].

Objective

Our objective as the FCC is to maximize the revenue taken from the winning bids. This is just a linear function of the valuations expressed by the bidders, and who won which segments. Toward this end, we define the objective:

$$\underset{x}{\operatorname{argmax}} \sum_{i\in B}\sum_{S\subseteq L} v_i(S)x_i(S)$$

Again, this objective looks like it's exponentially long to even write down! This is true, but only if done naïvely. We also need to worry about infeasible solutions to the objective (e.g., we don't want to assign every subset of segments to every bidder, which is clearly illegal), which is done next.

Constraints

There are only two constraints on the objective: (i) each bidder can win at most one of the conjunctions in her bid (the XOR constraint) and (ii) each wireless segment can be allocated to at most a single bidder. We can write these as follows (note that both sets of constraints are linear):

$$\sum_{S\subseteq L} x_i(S) \le 1 \forall i \in B$$

This states that the sum of the decision variables for each of the conjunctions for a specific bidder i can be either 0 (the bidder i won nothing) or 1 (the bidder i won exactly one conjunction).

$$\sum_{i\in B}\sum_{S\subseteq L \wedge j\in S} x_i(S) \le 1 \forall j \in L$$

This states that, for each individual wireless segment ℓ_j, the sum over all bidders of any conjunction that includes segment ℓ_j can be either 0 (nobody won this segment) or 1 (exactly one bidder won this segment).

Finding a solution to this integer linear program is (NP-)hard.However, you can quickly—i.e., in polynomial time—provide loose *upper* and *lower* bounds on the objective value of the integer program.

b) Formulate an *upper bound* for the problem, and prove that it is an upper bound. For this problem, please formulate an upper bound other than the LP relaxation. **5 pts**

—SOLUTIONS—

There are many possible upper bounds. For example, ∞ or $\sum_{i\in B}\sum_{S\subseteq L} v_i(S)x_i(S)$ are both extremely loose—that is, not close to the optimal feasible solution—bounds. Tighter bounds prune the search tree and prove optimality at leaves more quickly than looser bounds. The tradeoff here is finding a tight bound that is also relatively easy to compute (typically in "fast" polynomial time, like linear or quadratic). One such bound is:

$$\sum_{i\in B}\max_{S\in L} v_i(S)$$

That is, take the aggregate maximum valuations from each bidder. Since bidders can't have more than one conjunction accepted, this is clearly an upper bound. However, this may not be a *feasible* solution—but that's okay!

c) Formulate a *lower bound* for the problem, and prove that it is a lower bound. **5 pts**

—SOLUTIONS—

There are also many possible lower bounds. For example, 0 or $\min_{i\in B, S\subseteq L} v_i(S)$ are trivial, weak lower bounds. One nice (and common) lower bound is the greedy solution to the "packing problem", which is basically:

(a) Pick the bid with the highest valuation and allocate all its requested segments

(b) Repeat, only picking bids that do not intersect with those that have already been chosen, and are not owned by a bidder who has already won

By construction, this results in a feasible allocation of segments to bidders. Any feasible allocation is a lower bound on the optimal feasible allocation, so we're done. This is also easy to compute (worst-case quadratic, better with some smarts).

One way to solve the integer program defined above is by using a *branch-and-bound* tree search algorithm with A* node selection. This technique uses the divide-and-conquer paradigm to split the search problem into subproblems by assigning a value to each integer/binary variable at each node of the search tree.

d) Write a branch-and-bound solver that, given an instance of the FCC spectrum auction, solves the winner determination problem. This solver should use the upper and lower bounds you designed to prune the search tree at every node. **15 pts**

e) Compare the solution speed (runtime and search tree size) of your solver using (i) your upper bound and (ii) the LP relaxation of the subproblem at each node of the search tree. You can—and should—use a standard LP solver, many of which are available online. **10 pts**

—SOLUTIONS—

Instance	Objective
auction_m100_n100_p0.fcc	122
auction_m100_n100_p2.fcc	129.52
auction_m100_n200_p0.fcc	128
auction_m100_n200_p2.fcc	138.61
auction_m100_n20_p0.fcc	53
auction_m100_n20_p2.fcc	57.05
auction_m100_n50_p0.fcc	55
auction_m100_n50_p2.fcc	61.8
auction_m10_n10_p0.fcc	38
auction_m10_n10_p2.fcc	41.1
auction_m10_n20_p0.fcc	48
auction_m10_n20_p2.fcc	52.44
auction_m10_n2_p0.fcc	10
auction_m10_n2_p2.fcc	10.56
auction_m10_n5_p0.fcc	26
auction_m10_n5_p2.fcc	27.88
auction_m50_n100_p0.fcc	102
auction_m50_n100_p2.fcc	109.28
auction_m50_n10_p0.fcc	23
auction_m50_n10_p2.fcc	24.91
auction_m50_n25_p0.fcc	48
auction_m50_n25_p2.fcc	52.79
auction_m50_n50_p0.fcc	99
auction_m50_n50_p2.fcc	106.65

For most students, using an LP relaxation of the IP for an upper bound resulted in faster (in terms of time) solutions, and definitely lower node counts.

You can view code that writes the entire integer probgram above to a .lp format file and solves it via CPLEX here: `https://github.com/JohnDickerson`

Suggested LP solvers

Writing your own linear programming solver is a tricky task due to the ease in which numerical instability issues arise. We recommend you don't write your own solver for this homework; instead, check out one of the following:

Solver	Type	Download	Tutorial	Notes
CPLEX	Free (Academic)	Link	Link	Requires a (free) IBM Academic account
Gurobi	Free (Academic)	Link	Link	Requires a (free) Gurobi Academic account
CLP	Free	Link	Link	Very commonly used open source LP solver
SCIP (SoPlex)	Free	Link	Link	LP solver for the other big open source IP suite
lpsolve	Free	Link	Link	Slower than CLP/SoPlex, but the easiest to install

Note that many of these solvers can also directly solve integer linear programs. Please write your own branch-and-bound solver, as detailed above, but feel free to check your answers against any of these solvers (for correctness)!

Testing your solver

Your solver should accept auction data in the following form:

```
<num-segments>  <num-bidders>
<bidder-id>     <valuation>    <segment-id>  ...  <segment-id>
<bidder-id>     <valuation>    <segment-id>  ...  <segment-id>
     ⋮               ⋮               ⋮         ⋮      ⋮
<bidder-id>     <valuation>    <segment-id>  ...  <segment-id>
```

The first line represents m, the number of spectrum segments available for allocation and n, the number of bidders interested in the segments. The next lines specify one conjunction from a bidder's overall `XOR` bid. Note that everything is 1-indexed, not 0-indexed.

For example, a simple two-item auction with three bidders could like like this:

```
2  3
1  1.12  1
1  1.99  1  2
2  2.3   1
3  0.82  2
```

Here, bidder b_1 submitted bid $\langle(1.12, \ell_1) \oplus (1.99, \ell_1 \wedge \ell_2)\rangle$, while b_2 submitted bid $\langle(2.3, \ell_1)\rangle$ and b_3 submitted bid $\langle(0.82, \ell_2)\rangle$. The revenue maximizing allocation gives ℓ_1 to b_2 and ℓ_2 to b_3, for total revenue of 3.12.

Like with the last homework, we will generate some test instances and post them online. We will also release an instance generator with which you can more extensively test your code. These will be posted on the course website soon.

As output, your solver should write your Andrew ID and the revenue from a revenue maximizing allocation to standard out. That is, if you are Harry Q. Bovik with Andrew ID `hqbovik` and determine a revenue maximizing allocation with objective value of 12.355, your final output to standard out should be:

```
hqbovik  12.355
```

Bibliography

[1] BLUM, A., AND FURST, M. Fast planning through planning graph analysis. *Artificial Intelligence 90*, 1-2.

[2] LEHMANN, D., MÜLLER, R., AND SANDHOLM, T. The winner determination problem. *Combinatorial auctions* (2006), 297–317.

[3] VELOSO, M., PÉREZ, M., AND CARBONELL, J. Nonlinear planning with parallel resource allocation. In *Proceedings of the Workshop on Innovative Approaches to Planning, Scheduling and Control* (1990), Citeseer, pp. 207–212.