- all determinate literals found;[1] otherwise
- the literal with highest positive gain; otherwise
- the first literal considered that introduces a new variable.

We experimented with a number of different definitions of the *gain* function. We obtained the best performance using the "Laplace estimate" used in a number of recent learning systems (CN2 (Clark & Boswell, 1991), *etc.*):

$$\text{Gain}(r) = \frac{p+1}{p+n+2} \ ,$$

where r is the candidate rule, p is the number of positive examples covered by r, and n is the number of negative examples covered by r.

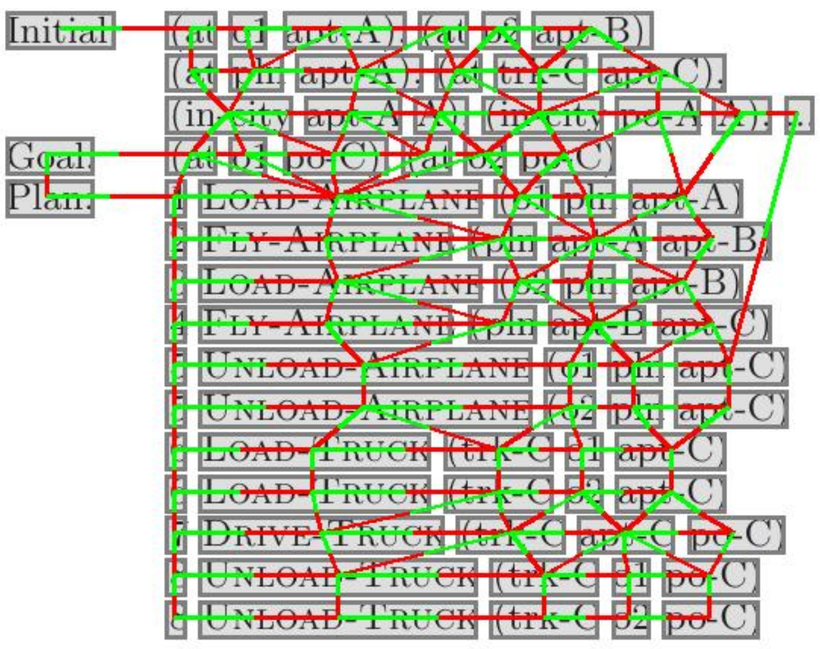
Because the Laplace estimate penalizes rules with low coverage, it proved to be more robust against noise in the training set. Noise is also handled, as noted earlier, by pruning learned rules that turn out to be inconsistent with future examples of solved planning problems given to the learner.

We mentioned above that during plan justification type inference on all objects and predicates is performed using the algorithm from TIM (Fox & Long, 1998). This inferred type information serves two purposes: First, types are used to reduce the number of rules considered in the learning procedure. For example, when considering candidate equality literals, the arguments of the equality literal must be of the same type. Second, when adding a literal that introduces new variables, inferred types are used to correctly find bindings for every new variable. For instance, when adding a literal (at o l) to a rule, where the types of o and l associated to the literal are "package"and "location" respectively and o is a new variable, only objects with type "package" will be considered as bindings for variable o. In other words, objects with type "truck" or "airplane" that can also be bound to the first argument of predicate (at o l) will not be considered. This use of type information is important to correctly acquire control rules.

It is common in inductive logic programming for the user to provide explicit background knowledge to the system in the form of additional relations or axioms (Quinlan, 1990). For example, the user might write a definition for a predicate that holds of "packages that need to be moved". One might argue, however, that manually defining good background knowledge is as difficult as defining good control rules. In our system, by contrast, no additional background knowledge is input by the user. The categorization of the different kinds of predicates, the state information that appears in the justified plan, and the inferred type information can all be considered to be kinds of background knowledge that is automatically acquired by the system.

[1] A *determinate* literal is one that introduces new variables so that there is exactly one binding for each positive example and at most one binding for each negative example in the partially-constructed rule (Muggleton & Feng, 1992; Quinlan, 1996).

Table 3. A simple logistics problem and its solution. There are three cities (A, B, and C), each containing two locations, an airport and a postoffice (*e.g.*, apt-A and po-A). In each city there is a truck (trk-A, trk-B, trk-C), and there is one airplane (pln). There are two packages (o1, o2) to be delivered.



2.4 Using Learned Rules

As reported in Huang, Selman, and Kautz (1999), we extended the PDDL planning input language (McDermott, 1998) of the Blackbox planner to allow control knowledge to be specified using temporal logic formulas. Thus the rules created by the learning module can be directly fed back to the planner. Blackbox uses static reject rules to prune the Boolean SAT encodings it creates of planning problems. All other kinds of rules are converted in propositional clauses that are added to the encoding. The general effect of the added clauses is to make the encoded problem easier to solve by increasing the power of the constraint propagation routines used by the system's SAT engines.

3. An Example

We will now step through an example of learning a control rule for the logistics domain. Consider a problem instance and the (justified) plan found by the Blackbox planner as shown in Table 3. Suppose we are learning static reject rules for the action "UNLOAD-AIRPLANE