Saving and reusing previously constructed plans is largely regarded as a promising approach to deal with the intractability of domain-independent planning (Hammond 1989; Kambhampati and Hendler 1992). But it has been shown that syntactically matching a new problem with a candidate case is NP-hard, and modifying a plan to suit a new problem can be strictly more difficult than generating a plan from scratch (Nebel and Koehler 1995). We present a case-based planning system that does not involve any plan modification and performs case matching very efficiently.

The system involves essentially a default planner, a plan library and learning from a set of training examples (cases solved by the default planner). To solve a new problem (with initial state $I_{\text{NEW}}$ and goal state $G_{\text{NEW}}$), a case (with initial state $I_R$, goal state $G_R$ and solved plan $P_R$) is retrieved from the library, such that the original task of finding a plan is reduced to that of finding two potentially simpler sub-plans. The first sub-plan $P_I$ transforms $I_{\text{NEW}}$ into $I_R$. The second sub-plan $P_G$ transforms $G_R$ into $G_{\text{NEW}}$. If an appropriate case is retrieved, $P_I$ and $P_G$ can be constructed by the default planner with less computational effort, and a concatenation of $P_I$, $P_R$ and $P_G$ constitutes a solution plan, as shown in the figure below. The most appropriate case is one where the predicted combined cost of developing $P_I$ and $P_G$ is the least, and significantly smaller than planning from scratch.

We used a neural network to learn how to predict the time the default planner would take to transform $I_{\text{NEW}}$ into $G_{\text{NEW}}$. $I_{\text{NEW}}$ into $I_R$ and $G_R$ into $G_{\text{NEW}}$. The predicted time taken is represented as a weighted combination of features of the problem. In our experiments, features were manually constructed, and both domain-dependent as well as domain-independent features were tested on.

Let $Distance(I,G)$ be the time the default planner is predicted to take to transform state $I$ into state $G$. Then the retrieved case $\langle I_R, G_R, P_R \rangle$ is the one with the highest $Gain$, defined by the following:

$$Gain = \frac{Distance(I_{\text{NEW}}, G_{\text{NEW}})}{Distance(I_{\text{NEW}}, I_R) + Distance(G_R, G_{\text{NEW}})}$$

$Gain$ can be computed very efficiently. Other target functions can also be learned (e.g. the quality of plan produced) and the $Gain$ metric can be accordingly defined in alternative ways.

A prototype of the system was implemented, with a STRIPS-based planning system developed by N. Short and L. Dickens as the default planner. A single-layer network was used for training and the system was tested in the blocks-world domain. A 25-35% improvement in planning performance was observed on the average for problems with 3-7 blocks. It is hypothesized that this method can be used to improve the average-case performance of other planners as well. The effectiveness of the system was found to depend heavily on how successful learning is, which relies primarily on the extraction of relevant problem features. Another determining factor is the library size. The overhead of searching a large library may exceed the gain obtained by reuse. This is the utility problem, which can be addressed in a number of ways (Minton 1985). The presented case-based framework has considerable potential to improve efficiency of planning in real-world domains, especially because it avoids plan modification and enables matching a problem with stored cases in a highly efficient way.

References


