State-Transition Cost Functions and an Application to Language Translation

Hiyan Alshawi   Adam L. Buchsbaum

AT&T Labs—Research
600 Mountain Ave.
Murray Hill NJ, 07974
{hiyan,alb}@research.att.com

September 5, 1996

Abstract

We define a general method for ranking the solutions of a search process by associating costs with equivalence classes of state transitions of the process. We show how the method can accommodate models based on probabilistic, discriminative, and distance cost functions and provide a means for assigning costs to unseen events that is applicable to all three models. By applying the method to our machine translation prototype, we are able to experiment with different cost functions and training procedures. These include the use of an unsupervised procedure for training the numerical parameters of our English-Chinese translation model. We present performance results from these experiments, showing that the choice of cost function leads to significant differences in translation quality.

AT&T Labs—Research   TR 96.11.1

1 Introduction

Our motivation for the work reported in this paper was the desire for a flexible way of assigning costs to translations produced by a speech translation system. Standard practice in the application of preference scores in machine translation (Brown et al. 1990; Chang, Luo, and Su, 1992; Sekine et al. 1992), and more generally in language processing for the purpose of disambiguation (Hindle and Rooth 1993; Resnik and Hearst 1993), has been to employ special-purpose training and scoring mechanisms. Our aim here is to take a more abstract view of the counts, weights, and penalties used for scoring in a search process and so provide a general costing mechanism that is independent of the details of the algorithms and information being applied.

In our translation prototype, costs are applied by the search components of the system to select between alternative translation derivations. An earlier version of our speech translation prototype lacked flexibility in this respect in two ways.

First, the costs were always interpreted as negated log probabilities of a statistical model of the process of deriving correct translations. This did not allow us to exploit all the data we had available to our training procedures. In particular, it did not use examples of "incorrect" translations, nor could it treat the correctness of solutions as a matter of degree, a clear deficiency in the case of translation.

Second, the particular statistical model for deriving translations was hard-wired into the mechanisms for counting events and computing and applying the log probabilities. This made it inconvenient to alter the model, for example to experiment with making some part of the model more "context dependent."

To gain the desired flexibility, we adopt an approach in which much of the mechanism required for handling costs is implemented as a separate module that is independent from its application to the translation problem. This facilitates experimentation with variants of our translation model, and with ways of associating costs with solutions, that are different from the standard probabilistic approach.

A particular example, for which we present experimental results, is an unsupervised method for associating costs with actions in a translation model. The structure of the model is assumed to be defined in advance, but the numerical costs account for a graded view of the "goodness" of a translation derivation. To do this we compute a distance metric between a source-language sentence and the result of translating it to the target language and back again. Cost penalties associated with "irregular" events leading to incomplete solutions are handled systematically in this framework.

In Section 2 we present the framework for associating costs with actions and solutions of a search process. We briefly review our translation model in Section 3 and discuss some of the cost functions we have tried for this application. Section 4 describes the training procedures used and the implementation of the costs module, and Section 5 presents results of experiments comparing some of the cost and training alternatives we have tried in the translation application.

2 Process Events and Costs

2.1 Costed Non-deterministic Processes

Our translation problem can be idealized in terms of *costed non-deterministic processes*. In its general form, such a process can perform multiple input and output operations. For the present paper, it is sufficient to restrict ourselves to simple costed non-deterministic processes, which have particularly simple input/output behavior. Specifically, a successful execution of the process consists of the following.

1. Performing a single input action: reading a string.

2. Executing a sequence of internal actions.

3. Performing a single output action: writing a pair consisting of a string and a real number.

4. Halting.

There is a *cost*, a real number, associated with each action. The output string is a *solution*, and the real number is the *cost* of that solution, defined as the sum of the costs for all actions performed before halting.

In the case of translation, the input string could be a source-language sentence (or an encoding of a speech signal or word lattice), the solution a possible translation into the target language, and the cost some measure of the degree to which meaning is distorted by this translation.

The process may have an arbitrarily complex internal state. Before performing its reading action, a simple process is in a unique initial state. This state is altered by the reading operation (except for a process that ignores its input) and (optionally) by each subsequent action taken by the process. For a given internal state, there is a set (not necessarily finite) of possible following actions to take, the *allowable* actions for that state. For a simple process, only the reading action is allowable in the initial state, and there are no allowable actions in the halting state. In a particular state, the next action taken is non-deterministically selected from the allowable actions. The cost for taking each action is determined by the state of the process before taking the action and by the particular action taken.

Given a characterization of a simple costed non-deterministic process, the standard search problem is to find a minimum-cost solution. For larger problems, it is also of practical importance to find a solution that has a high probability of having the lowest cost, or to find an approximate solution with a cost "close," in some sense, to the minimal cost. There are also situations in which we are interested in the N lowest-cost solutions for some integer N.

For the approximate search problem, a simple algorithm is to "run" the process repeatedly with the same input, keeping the lowest-cost solution observed. This approach is unrealistic for problems like speech recognition, language analysis, and translation, because the solution space is extremely large but only sparsely populated with low-cost solutions. For these problems, application-specific algorithms are devised to find good solutions quickly. Generally, these algorithms depend on techniques like dynamic programming to limit the search and/or share computation between different allowable execution paths. Our speech translation system makes extensive use of such techniques in various processing phases, but further discussion of algorithmic issues is beyond the scope of this paper.

2.2 Events and Contexts

In the previous section, we said that the cost of performing an action is determined by the internal state of the process before taking the action and by the action taken itself. For complex applications such as translation or speech recognition, it is only practical to assume that the cost depends only on certain aspects of the state and the action taken. (In current approaches to speech recognition, this corresponds to the Markovian assumption required to justify the use of hidden Markov models.)

We formalize this assumption by defining the cost of taking an action as a function of equivalence classes of states, which we will call *contexts*, and equivalence classes of actions, which we will call *events*. Here we further assume that contexts and events are representable by, that is, are in one-to-one correspondence with, finite sequences of symbols. (Of course, arbitrary finite data structures can be coded by such sequences.) We will often ignore the distinction between contexts or events and the symbol sequences used to represent them. For an event e and a context c we write

$$\begin{aligned} e &= < a_1, \ldots, a_n > ; \\ c &= < b_1, \ldots, b_m > . \end{aligned}$$

The alphabets from which the symbols a_i and b_j are taken may or may not differ and need not be finite.

As defined, the cost function specifying the costs of actions taken by a process can be represented by a function from pairs of symbol sequences to the real numbers. We borrow some notation from probability theory, writing the above event-context pair, which we call a *choice*, as

$$(e|c).$$

As an example, consider a simple process for non-deterministically applying a weighted finite state transducer to an input string, with the cost of a solution being the sum of the weights for the transducer arcs traversed. The cost function f for arc traversal choices would have the form

$$f((< b, s' > \mid < a, s >)) \mapsto w$$

where the arc goes from transducer state s to transducer state s' with input symbol a, output symbol b, and weight w. That is, we take as the context (an equivalence class of *process* states) the current transducer state together with the input symbol at the current position in the input string. The action taken would be to increment the current input position, add w to a cost accumulator, set the current state to s' and, "emit" b by, say, appending it to the end of an output string under construction.

The sets of allowable actions from states induces sets $A(c)$ of allowable events for each context c in the obvious way. Taken together, the events, contexts, specification of allowable events, and cost function constitute what we call a *process cost model*, or simply a *model*. The costs specified by the cost function are the *model parameters*; the other components constitute the *model structure*. Thus, the standard use of terms such as "context-dependent model" in speech recognition might correspond in our framework to a model in which the context symbol lists have been lengthened to distinguish further between context equivalence classes.

We contrast our costed event framework to the related effort toward providing a set of tools for processing weighted automata (Pereira, Riley, and Sproat 1994). The weighted-automata tools provide algorithms for manipulating labeled, weighted, state transition networks. In our approach, we further separate the algorithmic issues from those relating to cost manipulation. For example, the implementation of a speech recognizer using the weighted automata tools would be an instance of what we have called a costed process, and indeed the costs for arcs could be trained using our costed event model. Our framework, however, can be used in conjunction with any search process in which state transitions can be traced, regardless of the algorithms being applied by the process.

2.3 Probabilistic Models

Our use of terminology from statistical modeling, particularly from speech and language modeling, is not accidental. For a particular process, assume we have conditional probabilities

$$P(e|c)$$

expressing the probability that a process in a state represented by context c will perform an action represented by an event e, with the constraint

$$\sum_{e \in A(c)} P(e|c) = 1.$$

We can then define a process cost model with a cost function f, such as

$$f((e|c)) = -\ln(P(e|c)).$$

A minimum-cost solution with respect to this function will be a highest probability sequence of events performed by the process.

There may be external criteria for evaluating the utility of a solution proposed by a process. Initially, we consider this to be a binary decision, with some solutions judged positive and others negative. For a process implementing the translation problem, the process may produce target-language string solutions that preserve the meaning of the input to a variable extent or that fail to preserve it at all. For the purpose of a probabilistic model of translation, we assume that we have a judge (a human, or another system) who can identify pairings of source and target sentences as "correct" or "incorrect" translations. We can then estimate conditional probabilities based only on counts of events that lead to good translations in training against a corpus of source sentences. (Details of the training procedures we used in our experiments are described in Section 4.)

Given a set of solutions from executions of a process, let $n^+(e|c)$ be the number of times choice $(e|c)$ was taken leading to positive solutions and $n^+(c)$ be the number of times context c was encountered for these solutions. We can then estimate costs for a *standard probabilistic model* in the usual way:

$$f'((e|c)) = \ln(n^+(c)) - \ln(n^+(e|c)).$$

Assuming this estimate is exact, a minimum-cost solution for this model is one resulting from a highest probability sequence of events, *given* that the solution is positive.

2.4 Discriminative and Distance Models

One problem with the standard probabilistic model is that it disregards much of the information provided by the judgments made after running the process over some input strings. Specifically, it does not use counts associated with negative solutions. To remedy this, we have also used another model, the *discriminative model*. Positive counts $n^+(e|c)$ are derived from a set of executions as in the standard probabilistic model. Let $n^-(e|c)$ be the count for choice $(e|c)$ leading to negative solutions. The cost function for the discriminative model is

$$f'((e|c)) = \ln(n^-(e|c)) - \ln(n^+(e|c)).$$

This cost is an estimate of the negated log of the likelihood ratio comparing the probability, conditional on a context c, that the event e is taken on a path leading to a positive solution as opposed to being taken on a path leading to a negative solution. (See Dunning 1993 on the application of likelihood ratios in computational linguistics.)

Both the standard probabilistic model and the discriminative model ignore the degree to which a solution may be judged good or bad. This is clearly an issue in the case of translation, in which it is difficult to say that a translation is correct or incorrect in absolute terms. We have therefore experimented with another class of models that make use of an error or distance function. For the general case, we have a function h

$$h(s,t) \mapsto d$$

where s is an input string, t is a solution string, and d is a non-negative real number. The distance function is such that the greater the number it yields, the worse the solution is; it is zero for ideal solutions.

One simple type of model based on distance functions is the *mean distance model* (Alshawi and Carter 1994). Given a set of solutions resulting from executions of a process, the parameters of the mean distance model are

$$E_h(e|c),$$

the average of $h(s,t)$ for source-solution pairs s-t produced by a sequence of events including e in context c.

The mean distance model is still not ideal in that, compared with the probability model, it does not take into account that a particular choice faced by a process is always a choice between events with the same context. Another problem is that it is somewhat sensitive to peculiarities of the underlying distance function. To overcome these problems, we introduce the *relative distance model.* With the same assumptions we made for the mean distance model, let

$$E_h(c)$$

be the average of $h(s,t)$ for source-solution pairs s-t produced by a sequence of choices including the context c. The cost function f for the relative distance model is defined as

$$f(e|c) = \frac{E_h(e|c)}{E_h(c)}.$$

Thus a relative distance cost is the ratio of the expected distance for solutions involving an event in a particular context and the expected distance for solutions involving any allowable event for that context. With this cost function, "good" events, i.e., those corresponding to choices that tend to reduce the cost of a solution, have costs less than 1, whereas "bad" events have costs greater than 1. A cost of 1 indicates that the model is indifferent to the event, as compared to other available choices. This is useful in that it provides a specific neutral value to be assigned as the cost for unseen events.

2.5 Unseen Event Costs

A major issue in the application of statistical language models is the treatment of unseen events. For example, this issue has been addressed in the context of N-gram language models for speech recognition (Katz 1987; Jelinek, Mercer, and Roukos 1992), because it is not practical to obtain enough data to estimate the frequencies of all word trigrams. Our approach to deriving costs for unseen choices generalizes the notion behind the use of "backed off" likelihood estimates for N-gram language models. A similar technique has also been applied to the specific case of prepositional phrase attachment by Collins and Brooks (1995). (An alternative to this kind of approach would be the use of statistical or information theoretic techniques to cluster symbols in the event and context sequences along the lines of Pereira, Tishby, and Lee 1993.)

When faced with an unseen choice $(e|c)$ for which a cost is not available, the idea is to use instead the cost, suitably normalized, for $(e|c')$, where c' is a more general context than c, i.e., one representing a larger equivalence class of process states and therefore more likely to have been seen. It is also possible to apply a similar generalization to the equivalence class for the event e, but we have not yet tried this in our experiments.

There are many possible ways of generalizing the context c to a context c'. We have experimented with the following. The context in a choice

$$(e|c) = (a_1, \ldots, a_n | b_1, \ldots, b_m)$$

corresponds to a conjunction of conditions on process states s

$$p_1(s, b1) \wedge \cdots \wedge p_m(s, b_m)$$

for some binary predicates $p_1, \ldots, p_m$. We take c' as the equivalence class of contexts satisfying the less restrictive condition

$$\begin{array}{l}(p_2(s, b_2) \wedge \cdots \wedge p_m(s, b_m)) \vee \\ (p_1(s, b_1) \wedge p_3(s, b_3) \wedge \cdots \wedge p_m(s, b_m)) \vee \\ \vdots \\ (p_1(s, b_1) \wedge p_2(s, b_2) \wedge \cdots \wedge p_{m-1}(s, b_{m-1})),\end{array}$$

that is, a disjunction relaxing the conditions for c one at a time.

For the cost functions we presented earlier in this section, we can compute the cost for $(e|c')$ from prestored counts or distances for choices $(e|c_1), \ldots, (e|c_m)$, where the condition for context c_i is

$$p_1(s, b1) \wedge \cdots \wedge p_{i-1}(s, b_{i-1}) \wedge p_{i+1}(s, b_{i+1}) \wedge \cdots \wedge p_m(s, b_m).$$

The process can be applied recursively, if necessary, to an event context pair $(e|c_i)$, in case no instances of $(e|c_i)$ have been seen.

We refer to the cost corresponding to the logical conclusion of this process, i.e., the cost for $(e_0|c_0)$ where e_0 and c_0 are unrestricted event and context equivalence classes, as the *default cost* for the model. In particular, for models with relative distance costs, the default cost is 1, as noted earlier.

3 Application to Translation

3.1 Review of Translation Model

Our approach to translation follows the transfer paradigm. Monolingual analysis and generation components analyze and generate, respectively, dependency graphs like those employed in various forms of dependency grammar (Hudson 1984). These graphs express syntactic or semantic relations holding between the words of a sentence or phrase. The nodes of the graph are labeled with words and the arcs with relation symbols. A transfer component maps dependency graphs with source-language words into corresponding graphs with target-language words. The name "dependency" reflects the view that the basic structure of natural language phrases distinguishes one word as the "head" of the phrase (e.g., the main verb of a verb phrase) on which other subphrases (which in turn have heads) are "dependent."

We can divide the overall model applied by these components into the following model components.

- *Source-language monolingual model.* This is a costed model, the structure of which is specified by a lexicon associating with each word one or more "head machines," a particular kind of finite state machine that specifies the possible local dependency graphs headed by the word.

- *Target-language monolingual model.* The structure of this model is also specified by a lexicon associating head machines with target language words. In fact, the same head machine lexicon can be used for both analysis and generation of a particular language.

- *Bilingual transfer model.* The structure of this model is specified by a bilingual lexicon; each entry in the lexicon consists of a dependency subgraph in the neighborhood of a source word and its mapping to a corresponding target subgraph.

Head machine models and an algorithm for applying them efficiently in analysis, as well as an algorithm for efficiently applying the bilingual lexicon provided by the transfer model, are described by Alshawi and Buchsbaum (1996). Details of these algorithms are not relevant to this paper, which is limited to explaining how costs are associated with events of the underlying models applied by the algorithms.

The choices of the analysis model provide a good illustration of the structure of a costed model. Intuitively, when a head machine m is applied to analyze the dependents of a head word w_0, it starts with an incomplete phrase consisting only of w_0. Each state transition of m extends this phrase to the left or right by consuming a dependent phrase. When m stops, the phrase is considered complete. In the following choices, w_1 is a dependent of w_0 under relation r (w_1 is an *r-dependent* of w_0), and s and s' are states of head machine m (not to be confused with states of the entire analysis process).

- $(m|w_0)$. The dependent arcs with head w_0 may be constructed according to machine w_0.

- $(s|m, w_0)$. Machine m for head word w_0 may start in state s.

- $(\mathit{left}, s', r|m, s)$. In state s, machine m may construct an arc for a left r-dependent and enter state s'.

- $(\mathit{right}, s', r|m, s)$. In state s, machine m may construct an arc for a right r-dependent and enter state s'.

- $(w_1|w_0, r)$. An incomplete arc with label r from head w_0 may be completed by selecting a dependent subphrase headed by w_1.

- $(\mathit{stop}|m, s)$. Machine m may stop in state s.

The full state of the analysis process includes the heads and current extents of complete and incomplete phrases, the dependency trees associated with them, and other book-keeping information. It would thus be possible to make the model more fine grained by, for example, including the head word in the context of left, right, and stop events. (Linguistically, this corresponds to a graded rather than fixed model of subcategorization and modifiability.) There is, of course, a tradeoff between the granularity of the model and the difficulty of training model parameters with limited data.

3.2 Translation Distance Functions

Our analysis and translation models were originally conceived as probabilistic models. The translations selected by the search algorithms were those minimizing the sum of negated log probabilities in the standard way. We later experimented with a discriminative model to take into account examples of negative solutions, as explained in Section 4.2. After implementing the handling of costed events in a more general way, we experimented with distance-based models.

There have been a number of attempts at training probabilistic translation models from examples produced by human translators, notably automatically aligned corpora of parliamentary proceedings (Brown et al. 1990, Wu 1995). Given a set of pairs of translated examples, we can define a function for a distance-based model in our framework in terms of a string distance metric d between arbitrary sentences of the target language. The distance function h used for scoring possible translations t of a source sentence s would be

$$h(s,t) = d(t, t_0),$$

where t_0 is the human translation of s found in the corpus.

Ideally, the distance function d is very sensitive to differences that alter the meaning of the target sentence, somewhat sensitive to stylistic differences, and less sensitive to linguistic differences that have little effect on meaning or emphasis. Defining such an ideal distance metric is currently not a practical proposition. Instead we define a string distance metric that is somewhat superficial, accounting for the identity of the words in the two strings, word order measured in terms of bigrams, and the degree to which the translation was fragmented into phrases because translation into a complete sentence was not possible. Even this crude function is more appropriate to translation than, say, counting the number of insertions, deletions, and substitutions, because such a function would penalize phrase movement too heavily in cases like fronting a prepositional phrase.

If a translated corpus is not available, we can still apply such a distance function by retranslating the target string into the source language and measuring the distance between the resulting string and the original source. For this to be effective, the structure of the translation model, and that of a similar model for translating from the target language to the source, need to be reasonably constraining; otherwise, the parameters of the translation model will tend to prefer translations that mirror the source language structure. Translating from source to target and back to the source again can be treated as a single costed process in our framework. We refer to this as the *back-translation* process. The main advantage of training model parameters with the back-translation process is that it does not require any human supervision for identifying good translations or assigning scores to them.

For our English-Chinese air travel information application, we did not have a hand-translated corpus. We built the structures of the English and Chinese monolingual models, a collection of prototypical head machines for various parts of speech, together with some head machines for idiosyncratic words, by hand. We also built prototype local graph structures for bilingual transfer entries, together with idiosyncratic entries for translation idioms in the domain, by hand. This provided a sufficiently constraining model for us to train model parameters using a back-translation process.

3.3 Irregular Events

One problem frequently encountered in applying complex statistical language models is handling situations not covered by the statistical model itself. In our speech translation system, there are several types of such "irregular" situations, including missing vocabulary items and fragmentary analysis, transfer, and generation. Components produce fragmentary results when the structure of the model is too constraining to produce a complete, well-formed solution and heuristic methods are applied to produce partial solutions, for example the analysis of a sentence into two or more unrelated phrases. Since partial solutions are almost always less preferable than complete solutions, standard practice, which we followed in earlier versions of our system, is to assign high penalty costs to solutions resulting from the application of such robustness heuristics. Setting and updating the penalty values to achieve reasonable selection between competing partial solutions arising from different, often multiple, irregular events in the search process then becomes problematic.

In the costed process framework, we can simply treat irregular events like any other process choices and train cost parameters for them along with other parameters. There is thus no need to adjust penalty values manually. In some experiments, we retained one piece of a priori knowledge by multiplying the costs of all irregular events by a constant factor greater than unity. This constant was chosen to be sufficiently high to prefer regular solutions to irregular ones. The main benefit from automatically training the costs of irregular events (selecting between different irregular solutions) is not affected by the choice of this constant.

4 Computing Event Costs

4.1 Model Parameter Tables

To compute and retrieve event costs, we maintain a hash table H, indexed by choice. For any such choice $(e|c)$, the associated record in H is the triple[1]

$$H(e|c) = (n^{+}(e|c), n^{-}(e|c), C(e|c)).$$

For the probabilistic and discriminative models, $n^{+}(e|c)$ and $n^{-}(e|c)$ are the numbers of times that $(e|c)$ was seen leading to positive and negative solutions, respectively (for the probabilistic model, the negative counts are not used in the cost computation); for the distance model, $n^{-}(e|c)$ is the sum of the distances incurred by solutions derived in part by $(e|c)$, and $n^{+}(e|c)$ is the number of times $(e|c)$ led to one of those solutions. $C(e|c)$ is the cost for the associated choice, computed prior to run time with respect to the costing model desired.

Any instance of a process — for our purposes, a translation — induces a list of executed choices, which we call the *trace*. The above implementation allows efficient training and subsequent executions. During training, we simply modify the fields stored in the hash table appropriately for each choice contained in a trace; details are given in Section 4.2. During execution, the cost of a choice $(e|c)$ is immediately given by the corresponding hash table entry $H(e|c)$. If there is no such entry in H, we can either compute a backed off cost for $(e|c)$, based on the choices $(e|c')$ that have a relaxed context c' of c, as described in Section 2.5, or we can simply use the default cost for unseen events for the particular cost model in use.

Prior to execution, e.g., at load time, we compute the cost for each choice in the hash table. If we are going to compute backed off costs for unseen events, we first add to H entries for the choices with relaxed contexts of those choices already in H. That is, for each $(e|c)$ in H, we consider every c' that is a relaxation of c (see Section 2.5), and we augment the counts in H for $(e|c')$ by the corresponding values in $H(e|c)$. We then proceed to compute the costs for the choices in H.

For the discriminative model, this merely entails traversing the hash table H and, for each triple $(n^{+}(e|c), n^{-}(e|c), C(e|c))$, resetting

$$C(e|c) \leftarrow \ln n^{-}(e|c) - \ln n^{+}(e|c).$$

For the probabilistic and distance models, we first traverse H and precompute an ancillary hash table G, indexed by context. For the probabilistic model, for each context c,

$$G(c) = \sum_{e} n^{+}(e|c);$$

[1]We will generally abuse notation, denoting, for any hash table T, the record indexed in T by the key k as $T(k)$.

i.e., $G(c)$ records the number of times context c was observed leading to a positive result. Then we can traverse H and, for each triple $H(e|c) = (n^+(e|c), n^-(e|c), C(e|c))$, reset

$$C(e|c) \leftarrow \ln G(c) - \ln n^+(e|c).$$

For the distance model, for each context c,

$$G(c) = \frac{\sum_e n^-(e|c)}{\sum_e n^+(e|c)};$$

i.e., $G(c)$ records the average distance incurred by solutions leading to which context c was observed. Then we can traverse H and, for each triple $H(e|c) = (n^+(e|c), n^-(e|c), C(e|c))$, reset

$$C(e|c) \leftarrow \frac{n^-(e|c)/n^+(e|c)}{G(c)}.$$

The above implementation allows fast cost recording and retrieval during training and execution: for each choice in a trace, there is one corresponding hash table lookup for the choice if the choice was observed during training and, if the choice was not observed, one hash table lookup for each relaxed context of the choice, to compute the backed off cost (if backed off costs are being computed). The only time-consuming operation is cost computation, which is done only once, prior to execution (or any time that the cost model is changed). Furthermore, the implementation allows cost models to share the same training data when the data are consistent with the respective models. For example, the probabilistic and discriminative models use exactly the same training data; only the cost computation differs. To alternate between the models or experiment with new models that use the same type of data, one only has to recompute the costs in H, not reload the training data or, worse, retrain. The distance model uses a different set of training data.

4.2 Training Procedures

When training, we consider two hash tables as described in Section 4.1: the *runtime table* and the *training table*. The runtime table contains the parameter costs that the system retrieves when executing processes during training. The training routine itself modifies the parameter counts/distances that are stored in the training table. At some point during training, the training table might augment or replace the runtime table. We roughly characterize the various training methods that we have tried according to the following criteria.

1. Whether the training procedure is *supervised* or *unsupervised*.

2. Whether the training procedure starts with an empty runtime table (in which case, all costs retrieved are defaults for the particular model used) or a runtime table initialized with previous parameters.

3. Whether the runtime table is augmented or replaced by the training table at some point during the training procedure. If so, we say the training procedure is *iterative*; otherwise, it is *static*.

4. The type of data the training procedure gathers (e.g., counts, used by the probabilistic and discriminative models, or distances, used by the distance model).

We next discuss the supervised and unsupervised training variants we have employed for machine translation.

4.2.1 Supervised Training

The supervised training method we have used is as follows. We translate a set of sentences from the source language to the target language. A bilingual speaker then scores each translation (based on semantic and syntactic criteria). If a translation preserves meaning and grammar sufficiently for communicating between a person fluent in only the source language and a person fluent in only the target language, then it is *good*; otherwise, it is *bad*. We record the translations and associated scores and keep the information over time.

During training, we retranslate all the sentences for which we have scores. For each source sentence σ, let $\tau(\sigma)$ be the current translation of σ into the target language. If $\tau(\sigma)$ matches some previously scored translation $\tau'(\sigma)$, then for each choice $(e|c)$ in the trace of $\tau(\sigma)$, we increment the positive (negative) count for $(e|c)$ in the training table if $\tau'(\sigma)$ was good (bad). Note that we allow for multiple previously scored translations of σ. As the system evolves over time, older translations are no longer matched, so we must continually (re)translate and score new (and previous) sentences.

The above training procedure is suitable for the probabilistic and discriminative models.

4.2.2 Unsupervised Training

We use two unsupervised training procedures. We refer to the first method as the *fragmentation training* method. If a trace for a translation contains irregular events (cf. Section 3.3), then we call the corresponding translation *fragmented*. During fragmentation training, we translate each sentence σ in the input corpus into a corresponding $\tau(\sigma)$ in the target language. If $\tau(\sigma)$ is not fragmented, then we increment the positive counts in the training table for the choices in the corresponding trace; otherwise, we increment the respective negative counts. Furthermore, if $\tau(\sigma)$ is not fragmented, we repeat the process in the reverse direction, translating $\tau(\sigma)$ back into the source language. This training procedure is suitable for the probabilistic and discriminative models.

The second unsupervised training procedure we use is suitable for the distance model. We call it a *contrastive* method, because it compares the performance of two models (the runtime model and a perturbation of the same). Dagan and Engelson (1995) describe the application of such a technique to probabilistic classification problems.

During contrastive training, each sentence σ in the input corpus is translated twice, into $\tau_1(\sigma)$ and $\tau_2(\sigma)$; $\tau_1(\sigma)$ and $\tau_2(\sigma)$ are then retranslated back to the source language, resulting in $\tau(\tau_1(\sigma))$ and $\tau(\tau_2(\sigma))$. The translations $\tau_1(\sigma)$ and $\tau(\tau_1(\sigma))$ are computed using the runtime table to retrieve costs for choices; the translations $\tau_2(\sigma)$ and $\tau(\tau_2(\sigma))$ are computed using the runtime table some of the time and randomly chosen costs the rest of the time. Let T_1 be the combined traces of $\tau_1(\sigma)$ and $\tau(\tau_1(\sigma))$; similarly, let T_2 be the combined traces of $\tau_2(\sigma)$ and $\tau(\tau_2(\sigma))$. Let d be a string distance metric, as discussed in Section 3.2. Let $d_1 = d(\sigma, \tau(\tau_1(\sigma)))$ and $d_2 = d(\sigma, \tau(\tau_2(\sigma)))$. The contrastive training process increments the cumulative distances stored in the training table for the choices in $T_1 \backslash T_2$ by d_1; similarly, it increments the cumulative distances for the choices in $T_2 \backslash T_1$ by d_2. The process also increments the occurrence counts for each choice appropriately. The intuition is that the choices that both traces contain did not distinguish the divergent translations, whereas the choices that were contained in only one trace more directly led to the corresponding translation and thus are responsible for the distance between that translation and σ.

5 Experiments and Results

5.1 Atis Data

We initially split our corpus of approximately 19000 ATIS sentences into two sets: a training set of approximately 10000 sentences and a test set of approximately 9000 sentences. We used the initial training set to build and exercise our first-generation language models for the translation system; we never looked at sentences in the test set. Some of the training set was also scored as discussed in Section 4.2. Whenever we wanted to evaluate our system, we chose a set of (usually 100) sentences at random from the unseen test set, translated them, and scored the results. We then removed those sentences from the test set and added them to the training set.

Currently, the training set contains approximately 13000 of the ATIS sentences, and the remaining 6000 sentences comprise the test set. The entire training set is used as input to the unsupervised training procedures. Approximately 3200 of the sentences in the training set are scored. Of those 3200, approximately 1150 are currently matched during the supervised training process (800 matches are good, and 350 are bad). Additionally, we have hand tagged approximately 800 sentences from the training set to derive counts for prepositional phrase attachment events. We include these counts with the results of our supervised training procedure when evaluating our "supervised system."

5.2 Results

We performed various training experiments and evaluations on the resulting data.

1. We performed a supervised training experiment on the hand-scored ATIS sentences in our corpus. In effect, this was an iterative process, because we ran the training procedure using the results of previously collected supervised data, and so forth.

2. We performed an unsupervised, fragmentation training experiment on the 13000 ATIS sentences in our training corpus. This was also effectively an iterative process, because we used all our previously collected supervised data in the runtime table.

3. We performed an unsupervised, contrastive training experiment, using the 13000 ATIS sentences in our training corpus. We translated each sentence in the corpus five times: once using the runtime table, and four times each using the runtime table 75% of the time and randomly chosen costs 25% of the time. Thus, for each sentence, we had four contrastive pairings of traces: the "runtime trace" against each of the four "partially random traces." This experiment used an empty runtime table. (Thus all the runtime costs were defaults.) The results show the evaluation after one iteration through the training corpus. We discuss iterating this training procedure in Section 5.3.

4. We performed a variant of (3), populating the runtime table with the results of supervised training as well as the hand-tagged data.

Table 1 and Figure 1 show the results of evaluations on the above training data. For each test, we translated 200 test sentences. The first column of performance numbers are percentages of the test sentences that had good translations, as judged by a bilingual speaker; the second column represents percentages of the translations that preserved meaning, regardless of syntax errors. The qualitative number reflects the performance of the system with an empty runtime table; i.e., all choice costs are default, except that costs of irregular events are high enough to prefer unfragmented solutions,

Experiment	Meaning Preserved and Good Grammar	Meaning Preserved
Qualitative	29	71
Supervised		
Probabilistic	46	82
Discriminative	52	83
Unsupervised		
Empty runtime	37	71
Supervised runtime	54	83
Hybrid	56	83

Table 1: Results of evaluations on various training data.

and thus the model structure drives the system. The supervised numbers show the performance of the system using the results of supervised training as well as the hand-tagged data and hand-coded penalty costs, for both the probabilistic and discriminative cost models. The unsupervised numbers show the performance of the system using the results of contrastive training, in the context-normalized mean distance model, trained with the two different runtime tables described above. For the evaluation using the unsupervised data, we multiplied the costs of the irregular events by a large enough constant to prefer unfragmented solutions. The hybrid numbers show the performance of the system using the results of supervised and fragmentation training, the hand-tagged data, and the hand-coded irregular event costs, all costed under the discriminative model.

5.3 Interpretation of Data

We performed statistical tests to determine which training methods can be compared to one another based on the above data. The tests validate the following comparisons with respect to good translations.

- The supervised probabilistic and unsupervised (empty runtime) methods each outperform the qualitative method.
- The supervised discriminative, unsupervised (supervised runtime), and hybrid methods each outperform both the supervised probabilistic and unsupervised (empty runtime) methods.

Figure 2 depicts these comparisons.

It is worth noting that our best results do exploit supervised training in some way. On the other hand, bootstrapping off of initial supervised training data, the unsupervised contrastive method does improve the performance of the system. Continuing, that the qualitative method performs so well implies that the structures of our models themselves already encode a great deal of knowledge about the language. The differences between different costing models were much greater with the stricter evaluation test ("good translations" in Figure 1); thus the methods affected translation quality more than they did preservation of meaning. This may be partly an artifact of the ATIS domain, in which word sense ambiguity is relatively uncommon. As we move towards automatic learning of the model structure, we expect the differences between training methods to increase.

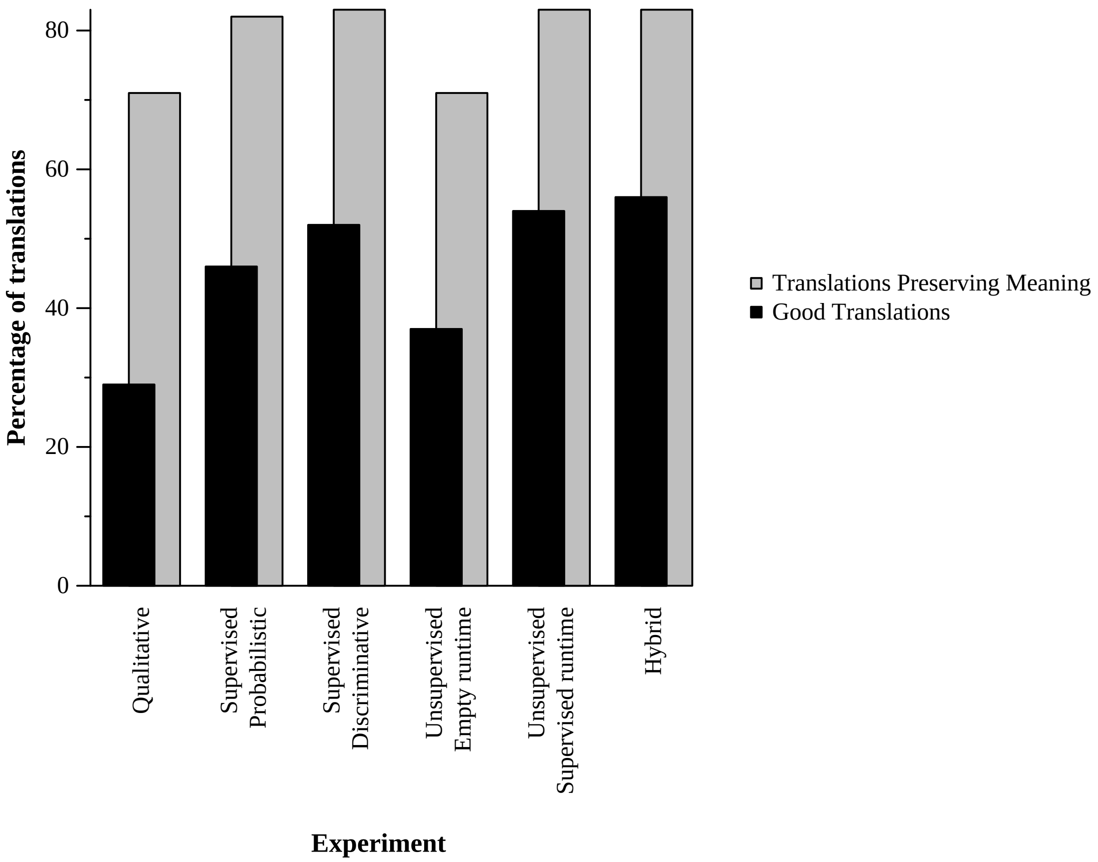

Figure 1: Results of evaluations on various training data.

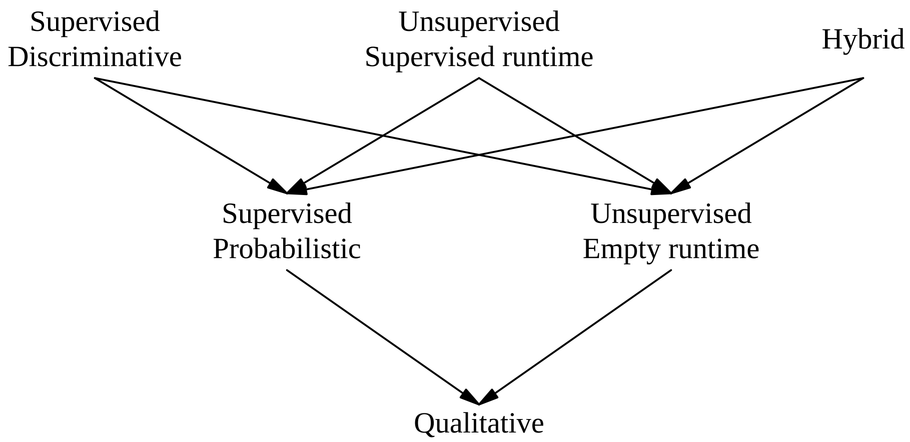

Figure 2: Statistically significant comparisons of training methods, with respect to good translations. An arc from method x to method y means that method x outperformed method y.

Finally, we have experimented with iterating the contrastive training method. While we do see some improvement in system performance from iterating, the actual data are not statistically significant.

6 Summary and Conclusion

We have presented a framework and implementation for associating costs with actions (state transitions) taken by a search process in order to rank the solutions proposed by the process. The costs are computed from parameters associated with process choices that correspond to equivalence classes of state transitions. Three cost functions were defined — probabilistic, discriminative, and distance — based respectively on log likelihoods, likelihood ratios, and distances in the solution space. We also provided a method, applicable to all three cost functions, for estimating costs for previously unseen events.

We then described the application of costed event models to automatic language translation, including the training of model parameters for the three cost functions. Experimental results on the performance of the different models were presented, showing in particular that all three cost models outperformed a baseline model and that the discriminative model outperformed the standard probabilistic model. The use of backed off costs for unseen events, however, did not significantly enhance performance compared to using uniform default costs in this application. The distance model provides a new approach to unsupervised training of translation models.

Overall, the implementation of a general event costing mechanism made it easier to experiment with different costing functions, abstracting this from the details of the translation task and the actual search algorithms used in translation. Since the use of costed event models only requires that state transitions of the underlying process are traceable, the framework is readily applicable to problems other than translation.

References

Alshawi, H. and A. L. Buchsbaum. 1996. "Head Automata and Tree Tiling: Translation with Minimal Representations". AT&T Labs—Research Tech. Report No. 96.10.1.

Alshawi, H. and D. Carter. 1994. "Training and Scaling Preference Functions for Disambiguation." *Computational Linguistics*, 20:635–648.

Brown, P., J. Cocke, S. Della Pietra, V. Della Pietra, F. Jelinek, J. Lafferty, R. Mercer and P. Rossin. 1990. "A Statistical Approach to Machine Translation." *Computational Linguistics* 16:79–85.

Chang, J., Y. Luo, and K. Su. 1992. "GPSM: A Generalized Probabilistic Semantic Model for Ambiguity Resolution." *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics*, ACL, 177–192.

Collins, M. and J. Brooks. 1995. "Prepositional Phrase Attachment through a Backed-Off Model." *Proceedings of the Third Workshop on Very Large Corpora*, Cambridge, Massachusetts, ACL, 27–38.

Dagan, I. and S. Engelson. 1995. "Committee-based sampling for training probabilistic classifiers." *Proceedings of ML-95, the Twelfth International Conference on Machine Learning.*

Dunning, T. 1993. "Accurate Methods for Statistics of Surprise and Coincidence." *Computational Linguistics*. 19:61–74.

Hindle, D. and M. Rooth. 1993. "Structural Ambiguity and Lexical Relations." *Computational Linguistics* 19:103–120.

Hudson, R. A. 1984. *Word Grammar*. Oxford: Blackwell.

Jelinek, F., R. L. Mercer and S. Roukos. 1992. "Principles of Lexical Language Modeling for Speech Recognition." In S. Furui and M. M. Sondhi (eds.), *Advances in Speech Signal Processing*, New York: Marcel Dekker, Inc.

Katz, S. 1987. "Estimation of Probabilities from Sparse Data for the Language Model Component of a Speech Recognizer." *IEEE Transactions on Acoustics, Speech, and Signal Processing* 35:400–401.

Pereira, F. C. N., M. Riley, and R. W. Sproat. 1994. "Weighted Rational Transductions and their Application to Human Language Processing." *Proceedings of the Human Language Technology Workshop*, Morgan Kaufmann, San Francisco, California, 262–267.

Pereira, F., N. Tishby and L. Lee. 1993. "Distributional Clustering of English Words." *Proceedings of the 31st meeting of the Association for Computational Linguistics*, ACL, 183–190.

Resnik, P. and M. A. Hearst. 1993. "Structural Ambiguity and Conceptual Relations." *Proceedings of the Workshop on Very Large Corpora*, ACL, 58–64.

Sekine, S., J. J. Carroll, S. Ananiadou, and J. Tsujii. 1992. "Automatic Learning for Semantic Collocation." *Proceedings of the Third Conference on Applied Natural Language Processing*, ACL, 104–110.

Wu, D. 1995. "Grammarless Extraction of Phrasal Translation Examples from Parallel Texts." *Proceedings of the Sixth International Conference on Theoretical and Methodological Issues in Machine Translation.* Centre for Computational Linguistics, Leuven, Belguim, 354–372.