Unimproved MT tuning and decoding

Jonathan Graehl

Better line search directions for MERT

- MERT picks optimal corpus-BLEU weights given an origin and a search direction (when decoding with pruning, this is approximate; redecode and merge forests or kbest lists until converged)
- When you have many features, there are many directions. Usually: orthogonal (vary only one feature) and a handful of random.
- Idea: pick better search directions

A better line search direction

- Use per-sentence BLEU (anti-)oracle+model score 1best (like MIRA hope+fear).
- Both model->hope and fear->hope directions seem reasonable (the direction is just the difference in the 1-best feature vectors)
- We can include as many directions as we want, so try both.
- It takes time to compute (anti-)oracles, so we randomly select batches of sentences and average the feature differences in each batch to generate a direction.

Objections?

- Why are we performing an expensive exact infinite-line search using a heuristic (local gradient inspired) direction?
- MERT has no smoothing. If you try to regularize the objective, then you lose the exact line search behavior.
- We need to still include random or orthogonal directions in case sparse features aren’t (yet) represented in model/hope/fear.

Does it work?

No.

(tuning) oracle BLEU weight	0 (baseline)	0.1	1	10	100
test BLEU	22.01	22.13	22.06	21.85	22.08

Tuning a urdu Hiero system with 10 dense features, there was no improvement. Convergence also wasn't any faster.

Orthogonal directions and random directions are used in all cases.

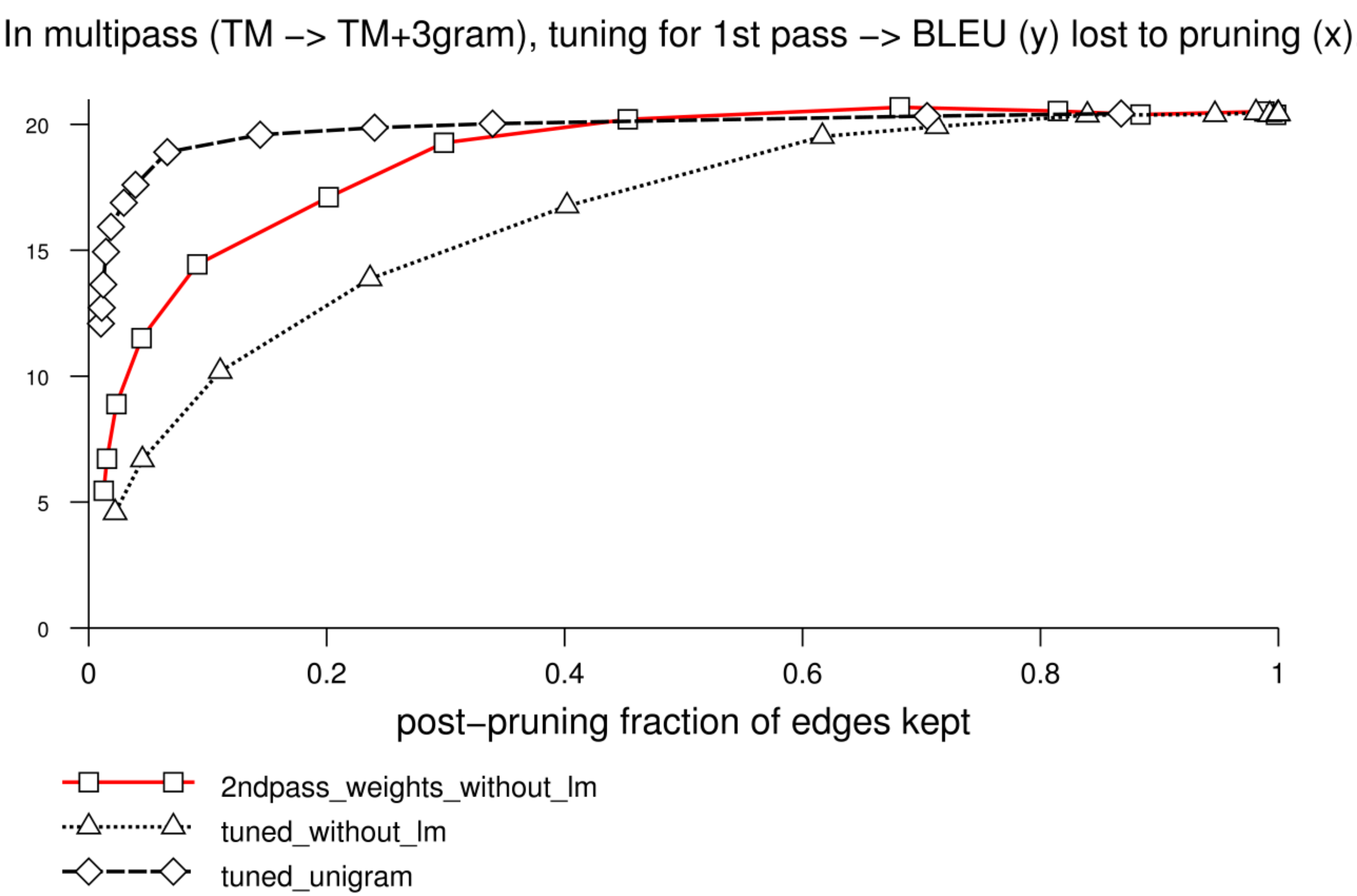
Since the preference for high model score vs. good or bad BLEU score is scale-dependent, I tried a wide range of weights.

Still possibly worth trying: more, sparser features. Different direction-averaging oracle batch sizes. More batches.

Possible confound: I didn't implement the hope/fear decoding myself; however, it's been verified to at least partially work: it yields kbests with better (hope) or worse (fear) BLEU.

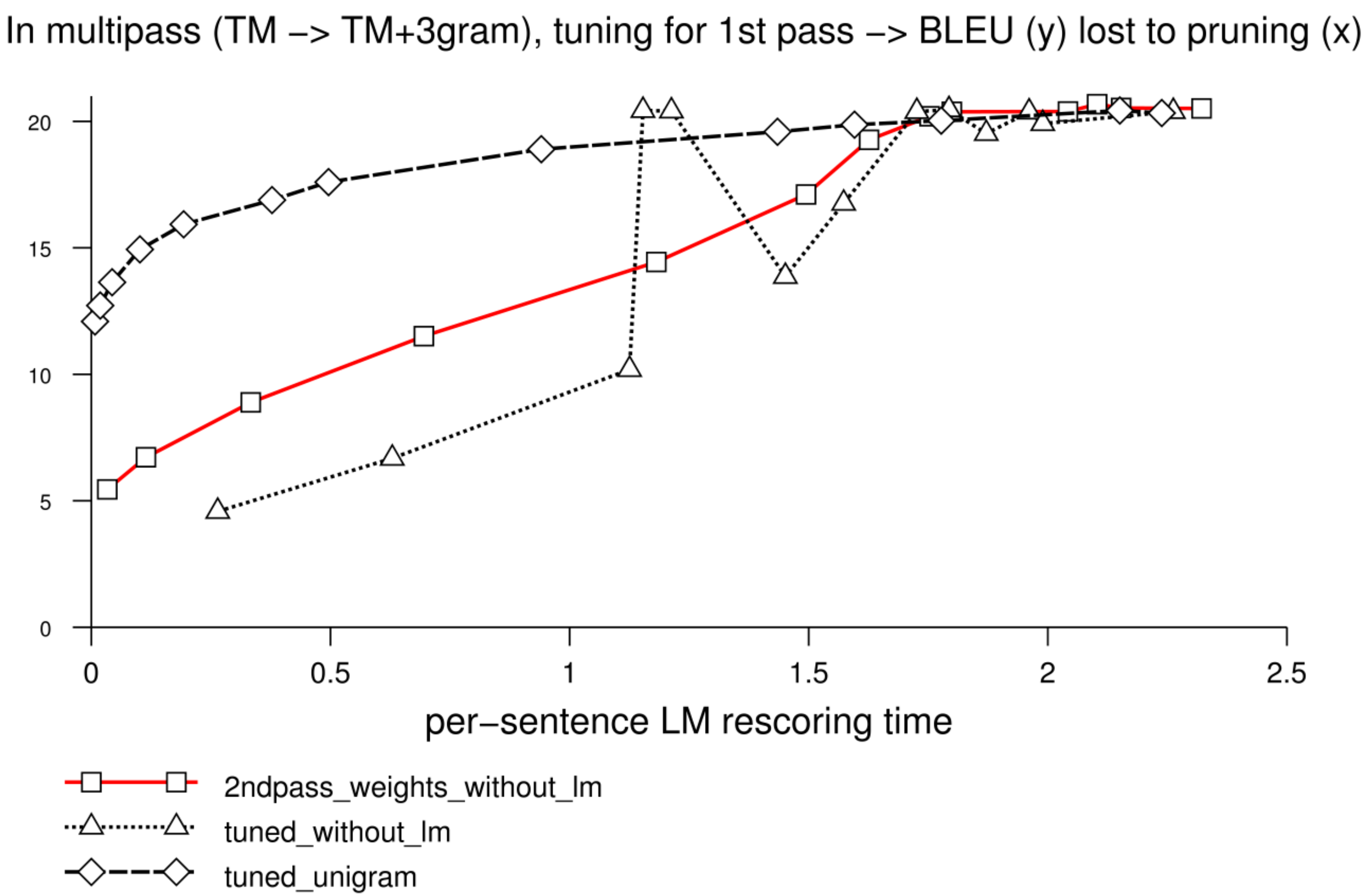
Faster LM rescoring of TM forests

We don’t need the whole TM forest to get the best translation using the ngram LM:



Is it faster?

Barely (if you want a good translation):



Soft pruning

- We lose too much good stuff by using an inside-outside global beam that removes large portions of the LM-unscored forest
- Coarse-to-fine using lower order ngrams or fewer bits per word (parts of speech or other classes) may work (but not as well as Petrov claimed)
- Idea: explore nodes that have poor without-LM model scores but only a little while cube pruning LM rescoring, varying the number of descendants explored smoothly (soft pruning, rather than 0% or 100% only).
- Baseline: N[i] = 200 if v[i]>threshold (per-word), 0 otherwise.
- Promise: N[i] ~ v[i]^β (normalized so average N[i]=200 or whatever)
- (N[i] is the number of cube pruning LM descendants explored for item i)
- (v[i] is the viterbi probability for item i: e^(λ•f) where f is the feature vector of the best derivation using i.)

Does it work?

- No.

β	0 (baseline)	0.01	0.1	10
BLEU	20.21	20.16	20.11	19.7
Ngram rescoring time (avg)	1.8s	1.9s	2.0s	1.1s

(Hiero Urdu 3gram)
Will it help with smaller baseline N[i] than 200?
Will it help with syntactic categories with/without per-span limits?

Thank you.