Mining Cinematic Discourse with Spectral Clustering

David R. Winer
School of Computing
University of Utah
Salt Lake City, UT 84102

April 17, 2017

Introduction

This project is about mining a corpus of similar movie scenes with shot-by-shot annotation. Since narratives such as film are ubiquitous, research in this area provides insights for extracting knowledge from a large space of existing resources.

The data set was created by a team of cognitive psychologists studying film comprehension. It contains 30 scenes all similarly depicting a Western style duel from Western films, mostly older ones. They annotated observations for each shot of each scene by denoting the type of action they observed, referring a dictionary of types they created for the data set, and they assigned entities in the scene to argument positions for those action types. They also denoted whether that action is starting and whether that action is finishing in the observation. These observations resemble literals that could be produced by a state-of-the-art computer vision system doing action recognition, or literals extracted from screenplay description text. However, there are mistakes throughout this data set, from spelling errors to argument positions being used inconsistently. Also, the choice of action types did not always seem intuitive, such as that some actions had overlapping meaning and others seemed to add interpretation to the what the characters were doing. One film was removed from evaluation because it had too many errors/abnormalities.

The goal is to use these observations to learn something about the scenes in the data set. The key strategy is to use a plan-based domain model to reconstruct the plot from these observations. I reviewed the most frequently used action types from their dictionary to created a planning domain model. Many of the actions were low-level, such as the motion of characters, which made this knowledge engineering feasible. However, simplifications were made, and many infrequent actions had to be discarded. In total, 32 plan operator types were used, either by combining or eliminating action types.

First, I introduce the key data structures used to compile observations and reconstruct the plot. Then I explain the precedure I used for spectral clustering entities. I also ran an evaluation using hand-coded roles I tagged some entities with in each scene. Finally, I introduce the action similarity measurement I used for agglomerative clustering.

Problem

Given an ordered set of scene observations, (1) reconstruct the scene's events, (2) cluster entities which have similar roles in story actions, and (3) cluster actions into discourse segments. Roughly speaking, the *similarity in roles* of two entities is the number of times those entities share the same argument position in an action type. Actions are clustered hierarchically using a measure of pointwise mutual information (PMI) used in similar work [CJ08; CJ09], modified to leverage that events which are in potential causal links are more likely to be related.

Data Structures

An action type designates a STRIPS-style declarative action model [FN72] which includes typed variable parameters, non-ground literal preconditions designating which conditions would need to hold for an instance of the action type

to occur, and non-ground effect conditions designating what conditions the action's execution would make hold. An example operators is in the appendix. An instance of an action is one where its variable parameters are substituted by consistent-typed entities.

Definition 1 (Action) *An **action instance** is a tuple $\langle t, V, a, P, E\rangle$ where t is an action type, V is an ordered list of entities, $a \in V \cup \emptyset$ is an agent which performs the agent, P is a set of ground function-free literal preconditions, and E is a set of ground function-free literal effects. If s is an action instance of the form $\langle t, V, a, P, E\rangle$, $eff(s) = E$, $pre(s) = P$, $agent(s) = a$, and $args(s) = V$.*

Action instances are observed via cinematic discourse in the format of camera shots. The same action instance can be observed in multiple shots. Intervals are represented as logical variables [AF94] (whose endpoints are millisecond values, *moments*, in $\mathbb{R}$). Let $\min(i)$ indicate the starting moment of i and $\max(i)$ denote the ending moment.

Definition 2 (Observation) *An **observed action** is literal of the form $obs(a, s, f, \tau)$ where a is an action instance, s is a boolean tag indicating whether the action begins at the observation, f is a boolean tag indicating whether the action finishes in the observation, and τ is an interval. If o is an observation of the form $obs(a, s, f, \tau)$ let $act(o)$ denote a, $ent(o) = args(act(o))$, $st(o) = s$, $fin(o) = f$, $\min(o)$ denotes $\min(\tau)$, and $\max(o)$ denotes $\max(\tau)$.*

Each scene is composed of shots, ordered chronologically, and each shot has a list of observed actions. I've included a figure in the appendix to visualize a snippet of how the observations are paired with camera shots for a scene (the actual corpus has no images).

An action instance can begin before the first time the action is observed, meaning the camera does not show the beginning of the action (or similarly does not show when the action finishes). Thus, we find the nearest observations with the same performing agent and infer that if the agent was observed and wasn't performing the action, then the action must have stopped/started at least at the beginning/end of this recent observation. Algorithm 1 steps through the inference procedure, which defines the **interval span** of an action instance.

Algorithm 1 `find-interval-span`

Input: Action observations A_{obs} and action instance a
Output: Interval span of a

$o_s := \arg\min_{o \in A_{obs}} (act(o) = a)$
$o_f := \arg\max_{o \in A_{obs}} (act(o) = a)$
$s, f = (\min(o_s), \max(o_f))$
$\Phi, \Omega = (st(o_s), fin(o_f))$
if $\neg\Phi$ **then**
    $s := \max_{o \in A_{obs}}$ s.t. $\max(o) < s, agent(act(o_s)) \in ent(o)$
end if
if $\neg\Omega$ **then**
    $f := \min_{o \in A_{obs}}$ s.t. $\min(o) > f, agent(act(o_f)) \in ent(o)$
end if
return (s, f) object with attributes $.s$ and $.f$

Definition 3 (Plot-Reconstruction) *A **plot-reconstruction** of a scene with action observations A_{obs} is a tuple of the form $\langle A, I, \prec, L, \rangle$ where A is a set of action instances for each unique action instance in A_{obs}, I is the `find-interval-span` function $I : A \to \mathbb{R}^2$ mapping action instances in A to their interval span, $\prec$ is an ordering over actions in A s.t. $a_i \prec a_j$ where $a_i, a_j \in A$ indicates that $I(a_i).f \leq I(a_j).s$, and L is a set of potential causal links of the form $a_i \xrightarrow{p} a_j$ where $a_i, a_j \in A$, $a_i \prec a_j$, $p \in eff(a_i) \cap pre(a_j)$, and $\neg\exists a_{threat} \in A$ s.t. $I(a_i).f \leq I(a_{threat}).f \leq I(a_j).f$) and $\neg p \in eff(a_{threat})$*

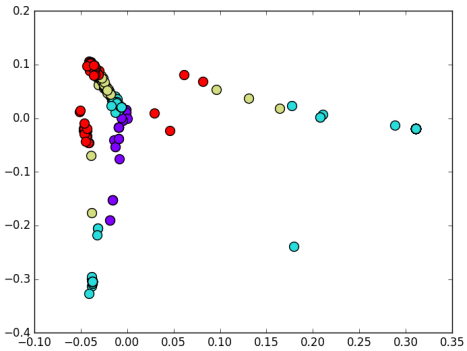
Entity Clustering

Entities are clustered across all scenes with an **entity matrix** - whose rows and columns are entities (256) and whose values are the number of distinct argument positions two entities co-occur in. I used spectral clustering with

Table 1: Entity Normalized Spectral Clustering

	muc	muc	muc	b^3	b^3	b^3
k	r	p	f	r	p	f
3	0.**9580**	0.**9605**	0.**9592**	0.**7870**	0.4441	0.**5678**
4	0.9389	0.9458	0.9423	0.6210	0.4420	0.5165
5	0.9237	0.9416	0.9325	0.5317	0.4505	0.4877
6	0.9046	0.9267	0.9155	0.4682	0.4527	0.4603
8	0.8817	0.9113	0.8962	0.4462	0.**4614**	0.4537

a normalized laplacian matrix. This matrix has weighted edges, so the degree matrix was the sum of the weights on its incident edges, and $L = I - D^{-\frac{1}{2}} W D^{-\frac{1}{2}}$ where W is the weighted entity matrix. Eigenvectors and eigenvalues are computed with numpy linalg module. I take the top k eigenvectors and scale them by $1/\sqrt{\lambda_i}$, then I transpose the matrix and compose 256 points in $\mathbb{R}^k$. I use the `gonzales` algorithm to find initial cluster centers and then run Lloyds to produce clusters. Below is a scatter plot of the points using just the top two dimensions of the $k = 5$ eigenvectors, when doing spectral clustering on entities.



A priori, I watched each scene and tagged some of the entities with one of several roles from set {dueler, observer duel-weapon, duel-location, amigo, thug, bartender, *untagged*}. These tags are in the project subfolder "entity_folder" and were used as the answer key chains in an evaluation. Two scoring metrics were implemented which are typically used for evaluating coreference resolution, MUC and B^3 [Rag+10].

The *MUC scoring metric* measures, in essence, how many times two entities which have the same role tag in the answer set are clustered together by spectral clustering. If $K_r = \{a, b, c, d\}$ is the set of entities hand tagged with role r, and $C_i = \{a, b\}$ is a cluster of entities, then let $p(K_r) = \{\{a, b\}, \{c\}, \{d\}\}$, a partitioning of K_r where elements are in the same subset if they are in some common cluster C'. *Recall* for K_r is measured as $|K_r| - |p(K_r)|/|K_r| - 1$, and *precision* is measured by swapping the roles of clusters and answer key sets.

The B^3 *scoring metric* computes recall for an entity e in an answer key set K by finding the cluster C containing e and incrementally adds $|K \cap C|/|K|$ for each entity, all divided by the number of entities, and computes precision for an entity e in a cluster C as $|C \cap K|/|C|$, incrementally added for all entities and divided by the number of entities.

Table 1 shows recall, precision, and f-score ($(2 * p * r)/(p + r)$) for the two scoring metrics for $k = 3, 4, 5, 6, 8$. Although the best results seems to come from $k = 3$, they all do reasonably well and I chose $k = 5$ moving forward to get a better segmentation of entities.

Action Clustering

Action types which are clustered may form meaningful story chunks reflecting different aspects of the scene type. These are of interest to use other features of the data set not in the scope of this work, such as attributes of the camera shot type, composition, and lighting. Although hierarchical clustering is slow, I only have 32 action types so the burden is bearable.

In previous work for event extraction from text [CJ08; CJ09; Hu+13] various forms of a pmi or pointwise

mutual information score is defined. I used the entity clusters from entity spectral clustering with $k = 5$ to label action instances by the number of times each action type had a particular entity cluster type in one of its argument positions. If $a_t(c)$ denotes action instances of type t with an arg mapping to cluster c let $count(t(c))$ indicate the number of instances of $a_t(c)$. Given a set of events $s(c), t(v)$,

$$pmi(s(c), t(v))) = log_2 \frac{P(s(c), t(v))}{P(s(c))P(t(v))}$$

where $P(s(c))$ is the probability that an instance of type s with arg in cluster c occurs:

$$P(s(c)) = \frac{count(a_s(c))}{\Sigma_t \Sigma_x count(a_t(x))}$$

and the joint probability (pmi numerator) is the number of times the two action types have args from the same cluster divided by all times two action types have args from the same cluster.

The results using this score for PMI were not promising and led to many unary clusters. I originally tried creating clusters for each action, arg position pair, similar to previous work which used verb and dependency slot [CJ08], but even for small k hierarchical clustering was too slow. Attempts to use spectral clustering also didn't create good clusters, where the weight of each edge in the adjacency matrix was their pmi. Later, I realized this was because unlike dependency in verbs, arg positions in different action types likely have no relationship.

Finally, I tried another measure which leverage the causal links between entities, where the joint probability is the percent of times two action types are in a potential causal link, and the denominator probabilities are the percent of times the action type is in any causal link. This score which I'll refer to as CP for causal potential outdoes pmi and $pmi + CP$ in terms of avoiding unary clusters for the clustering task. The distance measure between hierarchical clusters S, T with entity clusters in C was defined as:

$$dist(S, T) = \max_{s,t \in S \times T} \sum_{c \in C} pmi(s(c), t(c))$$

for complete-link and with $\min$ for single-link. The clustering procedure was to iteratively merge the two clusters with the largest minimum pmi between any two contained action types (single-link) or the largest maximum pmi between any two contained action types (complete-link). I've uploaded one example for $k = 5$ for single-link hierarchical clustering and an example $k = 6$ for complete-link in the appendix.

Conclusion

One of the main things I've learned, or learned to appreciate, is just how slow hierarchical clustering is (and that I should avoid it). For the results I was able to get, clusters seemed to make sense but also differed based on size of k. It's possible that the actions are not easily clustered and that it's easy to impose meaning onto clusters such as how they are related to each other, so it's difficult to draw any conclusions. Earlier I mentioned that I tried using spectral clustering where the matrix were argument positions for each action type, with degree weight as pmi, but then realized that this wasn't very meaningful. This caused me to not use spectral clustering for the action clustering, when really the problem was that arg positions in different action types are not meaningfully related, and it wasn't a problem with the clustering technique. If I had more time, I would try again to use spectral clustering for action types.

One reason that using action types which are in potential causal links lead to better results (in hierarchical clustering) is that actions which *can* form causal links are more likely constructed correctly and therefore are the best candidates for measuring event relatedness. If this is true, then it's not the causal relationships, and computing pmi with just the events which are in any potential causal link may also do well. Whether or not that's true, I'm also thinking that using just those actions in causal links would also be better candidates for clustering entities.

I'm looking forward to continuing this project and sharing this results with the cognitive psychologists who made the data set, whom I've collaborated with on other projects in the past. Learning clustering techniques was very useful for this data set and which I will use moving forward with my research in narrative intelligence.

References

[FN72] Richard E Fikes and Nils J Nilsson. "STRIPS: A new approach to the application of theorem proving to problem solving". In: *Artificial intelligence* 2.3 (1972), pp. 189–208.

[AF94] James F Allen and George Ferguson. "Actions and events in interval temporal logic". In: *Journal of logic and computation* 4.5 (1994), pp. 531–579.

[CJ08] Nathanael Chambers and Daniel Jurafsky. "Unsupervised Learning of Narrative Event Chains." In: *ACL*. Vol. 94305. Citeseer. 2008, pp. 789–797.

[CJ09] Nathanael Chambers and Dan Jurafsky. "Unsupervised learning of narrative schemas and their participants". In: *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*. Association for Computational Linguistics. 2009, pp. 602–610.

[Rag+10] Karthik Raghunathan et al. "A multi-pass sieve for coreference resolution". In: *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. 2010, pp. 492–501.

[Hu+13] Zhichao Hu et al. "Unsupervised Induction of Contingent Event Pairs from Film Scenes." In: *EMNLP*. Citeseer. 2013, pp. 369–379.

Appendix

obs:(draw-gun ['reno', 'gun01'], 1, 0)

obs:(draw-gun ['reno', 'gun01'], 0, 0)

obs:(draw-gun ['reno', 'gun01'], 0, 1)
obs:(fire-gun ['jed', 'gun02', 'reno'], 1, 0)

obs:(get-shot ['reno', 'reno-body'], 1, 0)

obs:(fire-gun ['jed', 'gun02', 'reno'], 0, 1)

obs:(get-shot ['reno', 'reno-body'], 0, 1)
obs:(falls ['reno'], 1, 0)

obs:(falls ['reno'], 0, 1)
obs:(dies ['reno'], 1, 1)

A scene snippet with annotated observations:

Below is the Western planning domain in PDDL (planning domain definition language) syntax:

```
(define (domain western-duel)
(:requirements)
(:types element char place item - object
observer bartender sheriff - char
bar - place
container - item
door - container
weapon - item
gun ammo tnt - weapon
horse bottle cig - item
step literal - element
)
(:predicates
   (has ?c - char ?it - item)
   (at ?object - object ?place - place)
   (= ?obj - object ?obj2 - object)
   (adj ?p1 - place ?p2 - place)
   (is ?literal - literal)
   (occurs ?step - step)

   ;guns
   (loaded ?g - gun)
   (loaded-with ?gun - gun ?ammo - ammo)
   (shot-at ?gun - gun ?target - object)
   (aimed-at ?g - gun ?target - object)
   (holstered ?g - gun)
   (cocked ?g - gun)
   (raised ?g - gun)
   (marksman ?c - char)
   (hit-by-bullet ?ob - object)
   (arrested ?c - char)

   ;cognitive
   (stare-at ?c - char ?ob - object)
   (sees ?c - char ?it - item)
   (facing ?c - char ?targ - object)
   (looking ?c - char ?targ - object)
   (staring ?c - char ?c2 - char)
   (provoked ?c - char)
   (bel-char ?c - char ?info - literal)
   (allies ?c1 - char ?c2 - char)
   (intends ?c - char ?goal - literal)

   ;physical states
   (alive ?char - char)
   (standing ?c1 - char)
   (on-ground ?c - char)
   (squared-off ?c - char ?c2 - char)
   (tied-up ?c - char)
   (drunk ?c - char)
```

```
   ;item specific
   (opened ?container - container)
   (closed ?container - container)
   (on-horse ?c - char ?h - horse)
   (smoking ?c - char)
   (drinking ?c - char)
   )

(:action leave
:parameters (?person - char ?p - place)
:precondition (and (alive ?person) (at ?person ?p))
:effect (not (at ?person ?p))
:agents (?person))

(:action arrive
:parameters (?person - char ?p - place)
:precondition (and (alive ?person))
:effect (at ?person ?p)
:agents (?person))

(:action run
:parameters (?person - char ?from - place ?to - place)
:precondition(and (at ?person ?from) (alive ?person))
:effect (and (not (at ?person ?from)) (at ?person ?to))
:agents (?person))

(:action arrest
:parameters (?sheriff - sheriff ?c - char ?p - place)
:precondition (and (at ?sheriff ?p) (at ?c ?p) (alive ?sheriff) (alive ?c))
:effect (and (arrested ?c))
:agents (?sheriff))

(:action stare-at
:parameters (?looker - char ?lookee - char ?loc - place)
:precondition(and
(at ?looker ?loc)
(at ?lookee ?loc)
(facing ?looker ?lookee)
(looking ?looker ?lookee)
(alive ?looker)
)
:effect (and (staring ?looker ?lookee))
:agents (?looker))

(:action look-at
:parameters (?looker - char ?lookee - char ?loc - place)
:precondition(and
(at ?looker ?loc)
(at ?lookee ?loc)
(facing ?looker ?lookee)
(alive ?looker)
)
:effect (and (looking ?looker ?lookee))
```

```
:agents (?looker))

(:action look-from-to
:parameters (?looker - char ?former-targ - object ?new-targ - object ?p - place)
:precondition (and (at ?looker ?p) (at ?former-targ ?p) (at ?new-targ ?p) (alive ?looker)
(looking ?looker ?former-targ))
:effect (and (looking ?looker ?new-targ) (not (looking ?looker ?former-targ)))
:agents (?looker))

(:action carry-from-to
:parameters (?carrier - char ?item - item ?p1 - place ?p2 - place)
:precondition (and (alive ?carrier) (at ?carrier ?p1) (at ?item ?p1))
:effect (and (at ?carrier ?p2) (not (at ?carrier ?p1)) (at ?item ?p2) (not (at ?item ?p1))
:agents (?carrier))

(:action face-at
:parameters (?facer - char ?facee - object)
:precondition(alive ?facer)
:effect (and (facing ?facer ?facee))
:agents (?facer))

(:action face-from-to
:parameters (?facer - char ?former-targ - object ?new-targ - object)
:precondition(and (alive ?facer) (facing ?facer ?former-targ))
:effect (and (facing ?facer ?new-targ) (not (facing ?facer ?former-targ)))
:agents (?facer))

(:action draw-gun
:parameters (?drawer - char ?gun - gun)
:precondition (and (has ?drawer ?gun) (holstered ?gun))
:effect (and (not (holstered ?gun)) (raised ?gun))
:agents (?drawer))

(:action raise-gun
:parameters (?c - char ?g - gun)
:precondition (and (has ?c ?g) (alive ?c) (not (holstered ?g)))
:effect (and (raised ?g))
:agents (?c))

(:action load-gun
:parameters (?c - char ?g - gun ?a - ammo)
:precondition (and (has ?c ?g) (has ?c ?a) (alive ?c))
:effect (and (loaded ?g) (not (has ?c ?a)))
:agents (?c))

(:action get-shot
:parameters (?victim - char ?gun - gun)
:precondition (and (shot-at ?gun ?victim))
:effect (and (hit-by-bullet ?victim))
:agents ())

(:action fall
:parameters (?faller - char)
```

```
:precondition (and (hit-by-bullet ?faller))
:effect (and (on-ground ?faller))
:agents()
)

(:action die
:parameters (?victim - char)
:precondition (hit-by-bullet ?victim)
:effect (not (alive ?victim))
:agents()
)

(:action walk
:parameters (?walker - char ?from - place ?to - place)
:precondition (and (at ?walker ?from) (adj ?from ?to) (not (on-ground ?walker)) (alive ?wa
:effect (and (at ?walker ?to))
:agents (?walker))

(:action mount
:parameters (?rider - char ?horse - horse ?p - place)
:precondition (and (alive ?rider) (at ?rider ?p) (at ?horse ?p))
:effect (on-horse ?rider ?horse)
:agents (?rider))

(:action dismount
:parameters (?rider - char ?horse - horse)
:precondition (and (alive ?rider) (on-horse ?rider ?horse))
:effect (and (not (on-horse ?rider ?horse)))
:agents (?rider))

(:action ride
:parameters (?rider - char ?horse - horse ?from - place ?to - place)
:precondition (and (on-horse ?rider ?horse) (at ?rider ?from))
:effect (and (at ?rider ?to) (at ?horse ?to))
:agents (?rider))

(:action reveal
:parameters (?revealer - char ?looker - char ?info - literal)
:precondition (and (is ?info) (alive ?revealer) (alive ?looker) (looking ?looker ?revealer
:effect (and (bel-char ?looker ?info))
:agents (?revealer))


(:action fire-gun
:parameters (?cowboy - char ?target - object ?gun - gun ?ammo - ammo ?loc - place)
:precondition(and (has ?cowboy ?gun)
  (at ?cowboy ?loc)
  (at ?target ?loc)
  (raised ?gun)
  (cocked ?gun)
  (aimed-at ?gun ?target)
  (loaded-with ?gun ?ammo)
  (alive ?cowboy)
```

```
  )
:effect (and
(shot-at ?gun ?target))
:agents (?cowboy))

(:action square-off
:parameters(?cowboy1 - char ?cowboy2 - char ?gun1 - gun ?gun2 - gun ?loc - place)
:precondition(and (has ?cowboy1 ?gun1)
  (has ?cowboy2 ?gun2)
  (alive ?cowboy1)
  (alive ?cowboy2)
  (at ?cowboy1 ?loc)
  (at ?cowboy2 ?loc)
  (staring ?cowboy1 ?cowboy2)
  (staring ?cowboy2 ?cowboy1)
  (holstered ?gun1)
  (holstered ?gun2)
)
:effect (squared-off ?cowboy1 ?cowboy2)
:agents (?cowboy1 ?cowboy2)
)

(:action provoke
:parameters (?provoker - char ?provokee - char ?loc - place)
:precondition (and  (at ?provoker ?loc)
(at ?provokee ?loc)
(alive ?provoker)
(alive ?provokee)
(not (squared-off ?provoker ?provokee)))
:effect (provoked ?provokee)
:agents (?provoker))

(:action adjust-clothing
:parameters (?adjuster - char)
:precondition (alive ?adjuster)
:effect ()
:agents (?adjuster))

(:action holster-gun
:parameters (?gunman - char ?gun - gun)
:precondition (and (has ?gunman ?gun) (alive ?gunman) (not (holstered ?gun)))
:effect (holstered ?gun)
:agents (?gunman))

(:action taunt
:parameters (?gunman1 - char ?gunman2 - char)
:precondition(and (squared-off ?gunman1 ?gunman2) (alive ?gunman1) (alive ?gunman2))
:effect (provoked ?gunman2)
:agents (?gunman1)
)

(:action side-step
:parameters (?side-stepper - char)
```

```
:precondition (alive ?side-stepper)
:effect ()
:agents (?side-stepper))

(:action smokes
:parameters (?smoker - char ?cig - cig)
:precondition(and (alive ?smoker) (has ?smoker ?cig))
:effect (smoking ?smoker)
:agents (?smoker))

(:action drinks
:parameters (?drinker - char ?bottle - bottle)
:precondition (and (alive ?drinker) (has ?drinker ?bottle))
:effect (drinking ?drinker)
:agents (?drinker))

(:action drink-with
:parameters (?drinker - char ?other - char ?p - place)
:precondition(and (alive ?drinker) (alive ?other) (drinking ?drinker) (at ?drinker ?p) (at
:effect (bel-char ?other (drunk ?drinker))
:agents(?drinker))

(:action assent
:parameters (?ger - char ?gee - char ?loc - place)
:precondition (and  (alive ?ger) (provoked ?ger) (provoked ?gee)
(alive ?gee)
(at ?ger ?loc)
(at ?gee ?loc))
:effect (and    (intends ?ger (squared-off ?ger ?gee))
(intends ?gee (squared-off ?ger ?gee))))

(:action de-escalate
:parameters (?c1 - char ?c2 - char)
:precondition (and (alive ?c1) (alive ?c2) (squared-off ?c1 ?c2))
:effect ()
:agents(?c1))

(:action open
:parameters (?c1 - char ?d - container ?p1 - place)
:precondition (and (alive ?c1) (at ?c1 ?p1) (at ?d ?p1) (not (opened ?d)))
:effect (opened ?d)
:agents (?c1))

(:action cheer
:parameters (?c1 - char)
:precondition (alive ?c1)
:effect ()
:agents (?c1))

(:action close
:parameters (?c1 - char ?d - container ?p1 - place)
:precondition (and (alive ?c1) (at ?c1 ?p1) (at ?d ?p1) (opened ?d))
:effect (closed ?d)
```

```
:agents (?c1))

(:action fall-from-to
:parameters (?faller - char ?from - place ?to - place)
:precondition (at ?faller ?from)
:effect (and (at ?faller ?to) (not (at ?faller ?from)) (on-ground ?faller))
:agents())

(:action ask-for
:parameters (?asker - char ?haser - char ?item - item ?p - place)
:precondition (and (alive ?asker) (alive ?haser) (has ?haser ?item)
(at ?haser ?p) (at ?asker ?p) (allies ?asker ?haser))
:effect (intends ?haser (has ?asker ?item))
:agents (?asker))

(:action identify
:parameters (?identifier - char ?recipient - char ?p - place)
:precondition (and (alive ?identifier) (at ?identifier ?p) (alive ?recipient) (at ?recipie
:effect (bel-char ?identifier (at ?recipient ?p))
:agents (?identifier))

(:action give
:parameters (?giver - char ?taker - char ?item - item ?p - place)
:precondition (and (alive ?giver) (alive ?taker) (has ?giver ?item)
(at ?giver ?p) (at ?taker ?p))
:effect (and (has ?taker ?item) (not (has ?giver ?item)))
:agents (?giver ?taker)
)

(:action take-gun
:parameters (?taker - char ?gun - gun ?takee - char ?p - place)
:precondition(and (alive ?taker) (has ?takee ?gun) (at ?taker ?p) (at ?takee ?P))
:effect (and (has ?taker ?gun) (not (has ?takee ?gun)))
:agents (?taker))

(:action aim-gun
:parameters (?c1 - char ?t - object ?p - place ?g - gun)
:precondition(and (alive ?c1) (at ?c1 ?p) (at ?t ?p) (has ?c1 ?g) (not (holstered ?g)))
:effect (and (raised ?g) (aimed-at ?g ?t))
:agents (?c1))

(:action stand-up
:parameters (?c1 - char)
:precondition (and (alive ?c1) (not (standing ?c1)))
:effect (and (standing ?c1) (not (on-ground ?c1)))
:agents(?c1))

(:action lower-gun
:parameters (?c1 - char ?g - gun)
:precondition (and (has ?c1 ?g) (raised ?g))
:effect (not (raised ?g))
:agents (?c1))
```

```
(:action drop-gun
:parameters (?c1 - char ?g - gun ?p - place)
:precondition(and (has ?c1 ?g) (at ?c1 ?p) (not (holstered ?g)))
:effect (and (not (has ?c1 ?g)) (at ?g ?p))
:agents (?c1))

; drop item, can't drop a horse though, which always has a location
(:action drop
:parameters (?c1 - char ?g - item ?p - place)
:precondition(and (has ?c1 ?g) (at ?c1 ?p) (not (at ?g ?p)))
:effect (and (not (has ?c1 ?g)) (at ?g ?p))
:agents (?c1))

(:action pick-up
:parameters (?c - char ?thing - item ?p - place)
:precondition (and (alive ?c) (at ?thing ?p) (at ?c ?p))
:effect (has ?c ?thing)
:agent (?c))

(:action pickup-gun
:parameters (?c1 - char ?g - gun ?p - place)
:precondition (and
(at ?g ?p) (at ?c1 ?p) (alive ?c1))
:effect (and
(has ?c1 ?g) (not (holstered ?g)))
:agents (?c1)
)

(:action offer-drink
:parameters (?c1 - bartender ?c2 - char ?p - place)
:precondition (and (alive ?c1) (alive ?c2) (at ?c1 ?p) (at ?c2 ?p))
:effect ()
:agents (?c1))

(:action help-up
:parameters (?c1 - char ?c2 - char ?p - place)
:precondition (and (alive ?c1) (alive ?c2) (at ?c1 ?p) (at ?c2 ?p) (not (standing ?c2)) (a
:effect (and (standing ?c2))
:agents (?c1 ?c2))

(:action cock-gun
:parameters (?c1 - char ?g - gun)
:precondition (and (has ?c1 ?g) (not (cocked ?g)) (alive ?c1) (not (holstered ?g)))
:effect (cocked ?g)
:agents (?c1))

(:action wince
:parameters (?c1 - char)
:precondition (provoked ?c1)
:effect ()
:agents ())
)
```

single-link
k=5

```
cluster:
drop
carry-from-to
fall
give
leave
cheer


cluster:
face-from-to
walk
raise-gun
die
pick-up
de-escalate
arrive
adjust-clothing
look-at


cluster:
run


cluster:
taunt
holster-gun
get-shot
lower-gun
fall-from-to
drop-gun
face-at
wince
reveal
draw-gun
identify
cock-gun
side-step
stand-up


cluster:
dismount
drink-with
```

complete-link
k=6

```
cluster:
leave
draw-gun
drink-with


cluster:
walk
get-shot
wince
side-step
face-at
give


cluster:
cheer
holster-gun
adjust-clothing
run


cluster:
taunt
pick-up
identify
arrive
lower-gun
dismount


cluster:
de-escalate
fall-from-to


cluster:
drop
raise-gun
reveal
stand-up
die
face-from-to
cock-gun
carry-from-to
fall
drop-gun
look-at
```

Figure 1: Example action type clusters for single link $k = 5$ and complete-link $k = 6$, both using entity clusters from spectral clustering with $k = 5$.

Another clustering of interest:

```
drop-gun
walk
die
raise-gun
fall


identify
dismount


arrive
wince
give
pick-up
draw-gun
cheer


drink-with
drop
carry-from-to
reveal
lower-gun
cock-gun
stand-up
side-step
adjust-clothing
face-at
look-at


fall-from-to
face-from-to


run
holster-gun
taunt
de-escalate
leave
get-shot
```