Research Contributions: Bit Flips

CS 197 | Stanford University | Arpita Singhal
cs197.stanford.edu

(Adapted from Prof. Michael Bernstein's slides)

Last time

Research introduces a fundamental new idea into the world

CS research can create sea changes in how we build computational systems and use them; these sea changes can drive major shifts in industry

CS research draws on many different methods — e.g., engineering, proof, design, probability, modeling — in different subfields

Administrivia

This week: form teams and start projects!

Weekly group project assignments start this week

Reminder: participation and attendance is factored into the grade

Today: bit flips and literature review

How do we get to the point where we know what has been done, and why our idea is different, new, and exciting?

We'll be using these skills in Assignment 2, due next week.

Your goal

Getting to a section of a paper that looks and feels like this →

Nominally, it surveys research; but, its true goal is to help you and others understand the novelty

Crowdsourcing is the process of making an open call for contributions to a large group of people online [7, 37]. In this paper, we focus especially on *crowd work* [42] (e.g., Amazon Mechanical Turk, Upwork), in which contributors are paid for their efforts. Current crowd work techniques are designed for decomposable tasks that are coordinated by workflows and algorithms [55]. These techniques allow for open-call recruitment at massive scale [67] and have achieved success in modularizable goals such as copyediting [6], real-time tran-

RELATED WORK

In this section, we motivate flash organizations through an integration of the crowdsourcing and organizational design research literature, and connect their design to lessons from distributed work and peer production (Table 1).

Crowdsourcing workflows

Crowdsourcing is the process of making an open call for contributions to a large group of people online [7, 37]. In this paper, we focus especially on *crowd work* [42] (e.g., Amazon Mechanical Turk, Upwork), in which contributors are paid for their efforts. Current crowd work techniques are designed for decomposable tasks that are coordinated by workflows and algorithms [55]. These techniques allow for open-call recruitment at massive scale [67] and have achieved success in modularizable goals such as copyediting [6], real-time transcription [47], and robotics [48]. The workflows can be optimized at runtime among a predefined set of activities [16]. Some even enable collaborative, decentralized coordination instead of step-by-step instructions [46, 86]. As the area advanced, it began to make progress in achieving significantly more complex and interdependent goals [43], such as knowledge aggregation [30], writing [43, 61, 78], ideation [84, 85], clustering [12], and programming [11, 50].

One major challenge to achieving complex goals has been that microtask workflows struggle when the crowd must define new behaviors as work progresses [43, 44]. If crowd workers cannot be given plans in advance, they must form such action plans themselves [51]. However, workers do not always have the context needed to author correct new behaviors [12, 81], resulting in inconsistent or illogical changes that fall short of the intended outcome [44].

Recent work instead sought to achieve complex goals by moving from microtask workers to expert workers. Such systems now support user interface prototyping [70], question-answering and debugging for software engineers [11, 22, 50], worker management [28, 45], remote writing tasks [61], and skill training [77]. For example, flash teams demonstrated that expert workflows can achieve far more complex goals than can be accomplished using microtask workflows [70]. We in fact piloted the current study using the flash teams approach, but the flash teams kept failing at complex and open-ended goals because these goals could not be fully decomposed a priori. We realized that flash teams, like other crowdsourcing approaches, still relied on immutable workflows akin to an assembly line. They always used the same pre-specified sequence of tasks, roles, and dependencies.

Rather than structuring crowds like assembly lines, flash organizations structure crowds like organizations. This perspective implies major design differences from flash teams. First, workers no longer rely on a workflow to know what to do; instead, a centralized hierarchy enables more flexible, de-individuated coordination without pre-specifying all workers' behaviors. Second, flash teams are restricted to fixed tasks, roles, and dependencies, whereas flash organizations introduce a pull request model that enables them to fully reconfigure any organizational structure enabling open-ended adaptation that flash teams cannot achieve. Third, whereas flash teams hire the entire team at once in the beginning, flash organizations' adaptation means the role structure changes throughout the project, requiring on-demand hiring and onboarding. Taken together, these affordances enable flash organizations to scale to much larger sizes than flash teams, and to accomplish more complex and open-ended goals. So, while flash teams' predefined workflows enable automation and optimization, flash organizations enable open-ended adaptation.

Organizational design and distributed work

Flash organizations draw on and extend principles from organizational theory. Organizational design research theorizes how a set of customized organizational structures enable coordination [52]. These structures establish (1) roles that encode the work responsibilities of individual actors [41], (2) groupings of individuals (such as teams) that support local problem-solving and interdependent work [13, 29], and (3) hierarchies that support the aggregation of information and broad communication of centralized decisions [15, 87]. Flash organizations computationally represent these structures, which allows them to be visualized and edited, and uses them to guide work and hire workers. Some organizational designs (e.g., holacracy) are beginning to computationally embed organizational structures, but flash organizations are the first centralized organizations that exist entirely online, with no offline complement. Organizational theory also describes how employees and employers are typically matched through the employee's network [23], taking on average three weeks for an organization to hire [17]. Flash organizations use open-calls to online labor markets to recruit interested workers on-demand, which differs dramatically from traditional organizations and requires different design choices and coordination mechanisms.

Organizational design research also provides important insight into virtual and distributed teams. Many of the features afforded by collocated work, such as information exchange [64] and shared context [14], are difficult to replicate in distributed and online environments. Challenges arise due to language and cultural barriers [62, 34], incompatible time zones [65, 68], and misaligned incentives [26, 66]. Flash organizations must design for these issues, especially because the workers will not have met before. We designed our system using best practices for virtual coordination, such as loosely coupled work structures [35, 64], situational awareness [20, 27], current state visualization [10, 57], and rich communication tools [64].

Peer production

Flash organizations also relate to peer production [3]. Peer production has produced notable successes in Wikipedia and in free and open source software. One of the main differences between flash organizations and peer production is whether idea conception, decision rights, and task execution are centralized or decentralized. Centralization, for example through a leadership hierarchy, supports tightly integrated work [15, 87]; decentralization, as in wiki software, supports more loosely coupled work. Peer production tends to be decentralized, which offers many benefits, but does not easily support integration across modules [4, 33], limiting the complexity of the resulting work [3]. Flash organizations, in contrast, use centralized structures to achieve integrated planning and coordination,

The Bit Flip

Recall: novelty

If the idea is already in the world, it is not considered **novel**, and thus not research.

In other words, to do research, you need to achieve something that nobody else has ever done. That novel achievement is called the **contribution** of your research.

You'll hear people say things like:

> "This is an extremely novel contribution."
>
> "This work is a tad too incremental." (its improvement or level of creativity over the state of the art is only minimal)

Eureka!

Novel ideas rarely spring forth fully formed from a researcher's head. They're not cool ideas that erupt out of the void.

They're much more often pivoted off of today's work:

- Some constraint that exists but shouldn't, or vice versa
- A realization that an idea has been applied in domains like X and needs to be rethought in domains like ~X
- A recognition that others have tried this technique in users of context A, or data of up to size N, but ~A or >>N breaks the technique.

In other words, research ideas arise as a reaction to the researcher's understanding of how people think about the problem today.

Bit flip: invert an assumption

Those examples were instances of a **bit flip**: an inversion of an assumption that the world has about how the world is supposed to work.

Recipe for a bit flip:

1) Articulate an assumption, often left implicit in prior work: this is the bit

2) **"No, it should be this way instead:"** argue for an alternative to that assumption

Bit	Flip	Project
Network behaviors are defined in hardware, statically.	If we define the behaviors in software, networks can become more dynamic and more easily debuggable.	Software-defined networking
Code compilers should utilize smart algorithms to optimize into machine code.	Code compilers will find more efficient outcomes if they just do monte carlo (random!) explorations of optimizations.	STOKE
A minimum graph cut algorithm should always return the optimal answer.	A randomized, probabilistic algorithm will be much faster, and we can still prove a limited probability of an error.	Karger's algorithm

Bit	Flip	Project
Activity tracking requires custom hardware.	Activity tracking requires just a standard cell phone.	Ubifit
NLP machine learning models should read sentences word by word, so the model can see what's before the current word	NLP machine learning models should consume the entire sentence at once, so the parser can see what's before and after	BERT

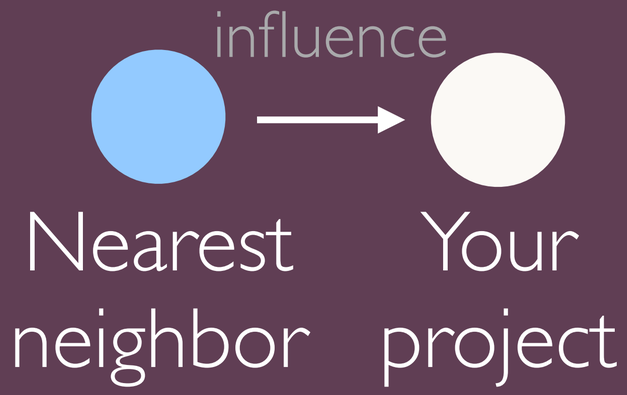
Single paper bit flip

Your TA is starting your team out on the project with a paper that is adjacent to your idea. Think of this as your **nearest neighbor paper**.

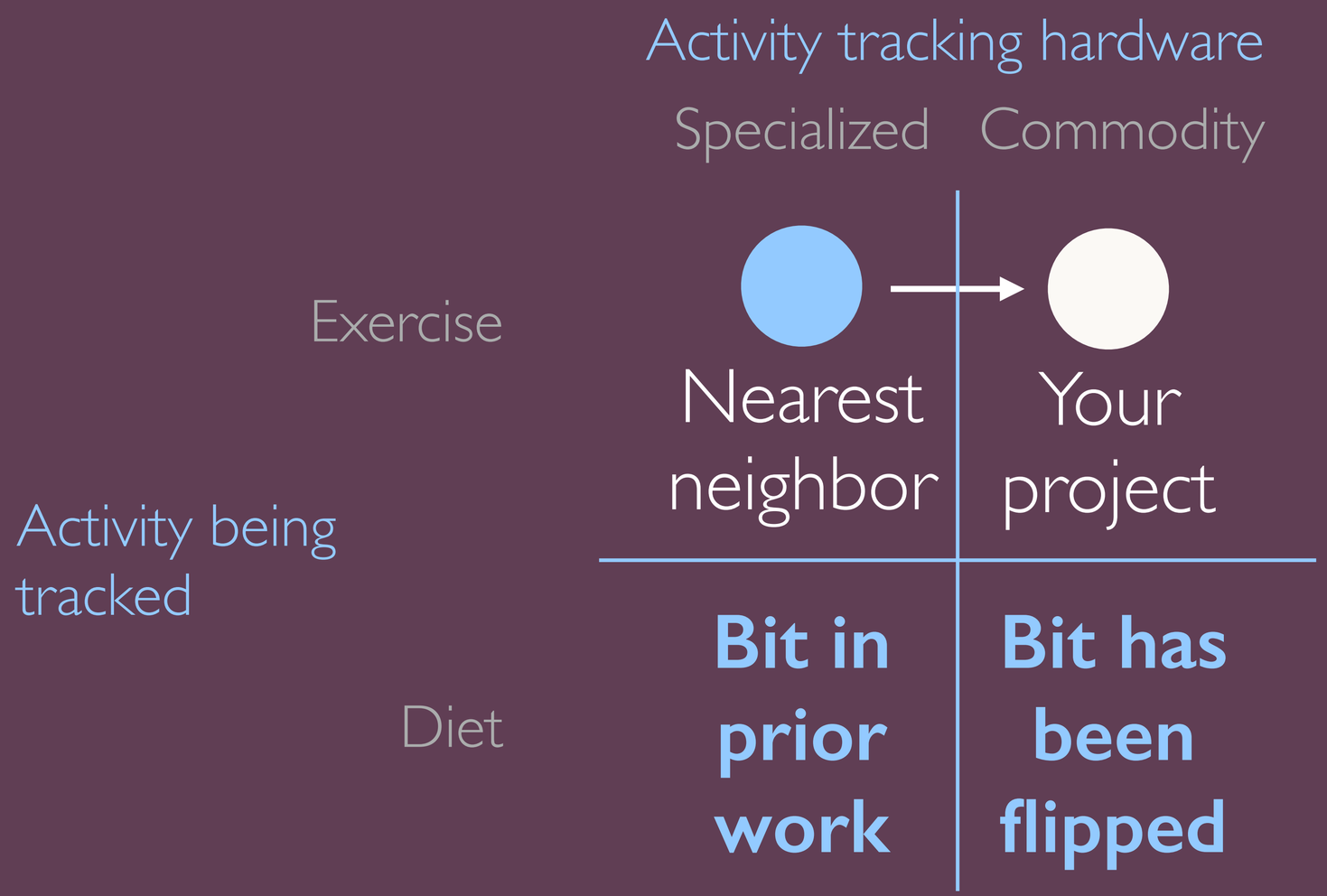
Your paper will be some sort of delta off of that paper.

- What assumption or limitation did it have, that you're flipping?

Literature search graph



Literature search graph

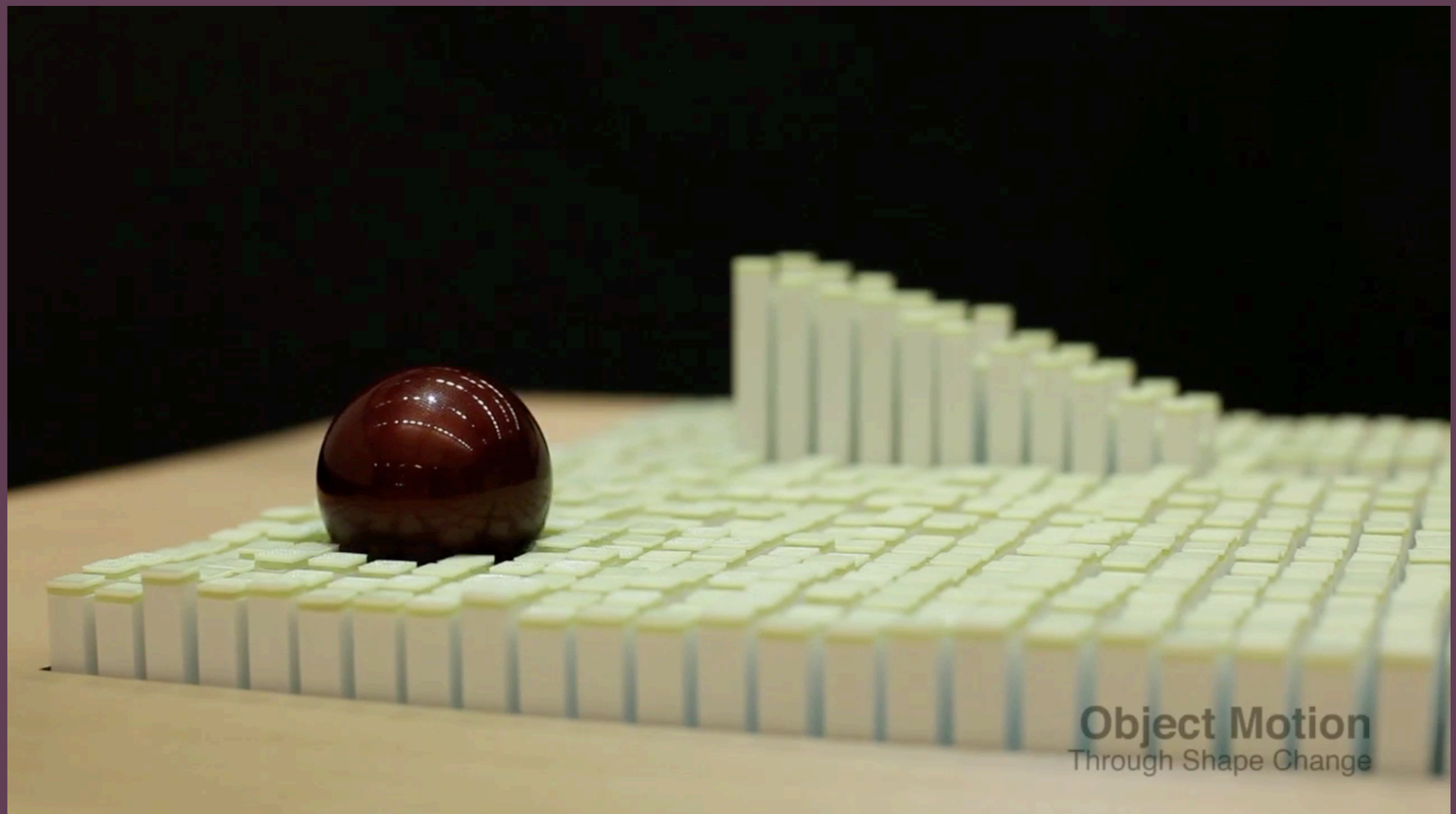


Imagine a set of design axes.

Your project should maintain position on most of them, but differentiate itself along one axis.

Single paper bit flip: exercise

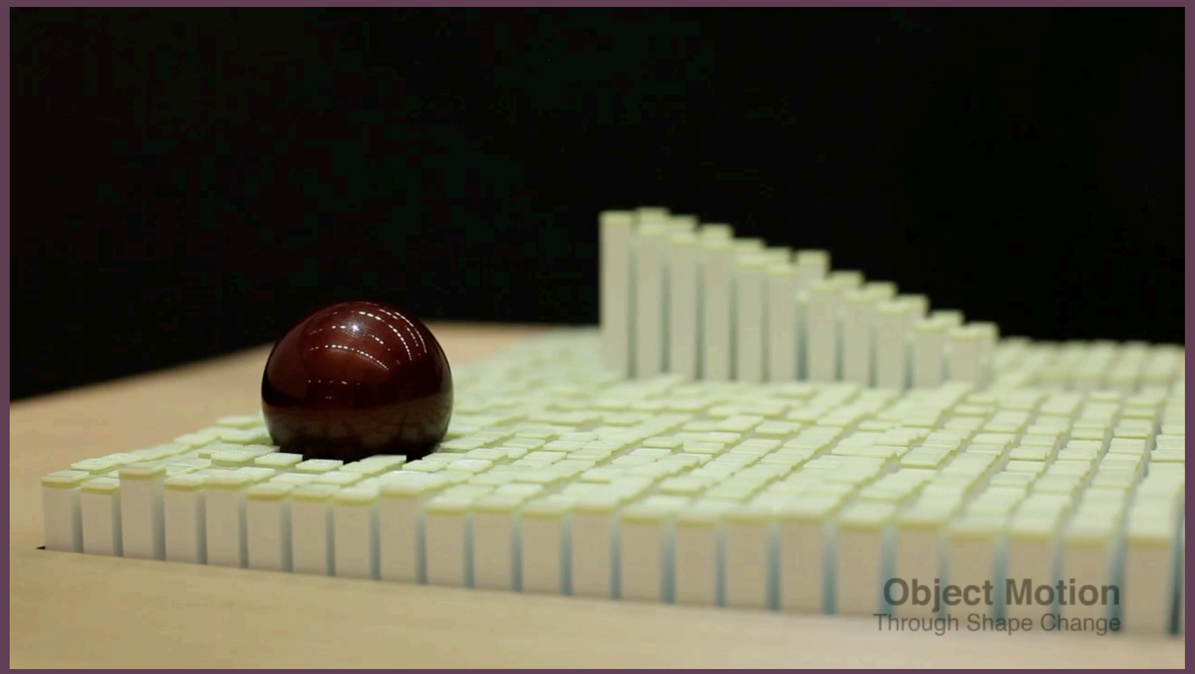
Nearest neighbor paper [Follmer et al. ’13] :



Single paper bit flip: exercise

Your idea: manipulators with small mobile robots [Le Goc et al.‘16] :

What's the bit flip? [1 min]



Literature-level bit flip

Eventually your goal is not to pivot off a single paper, but off the literature more broadly. This makes for a stronger argument of novelty.

Recipe:

1) Read the literature. (Which papers? Stay tuned…)

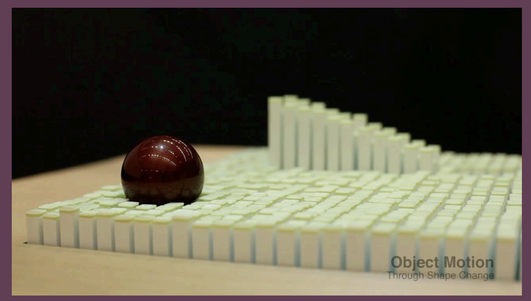
2) What assumptions underlie all of the papers? $\forall p \in$ papers: …

3) Which assumption are you changing? And why does it matter to the literature?

Literature-level bit flip

Why do a literature-level bit flip instead of a paper-level bit flip?

There exist many possible bit flips for a single paper:

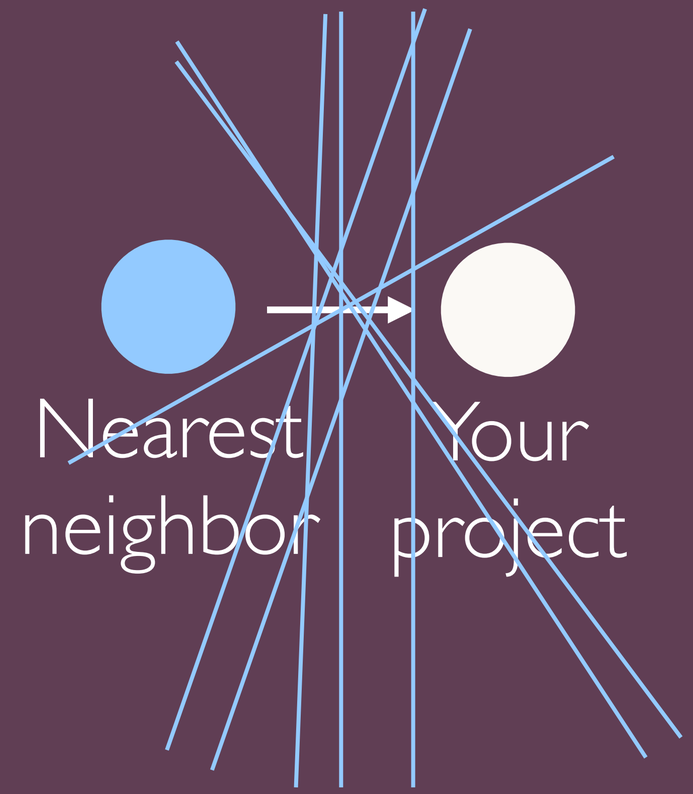


What if we changed the size of the pins?
What if we vibrated the pins for haptic feedback?
Could we make it mobile rather than mounted on a table?

…but not every possible bit flip matters. Some are incremental.

You identify more important ideas by bit flipping across a broader literature.

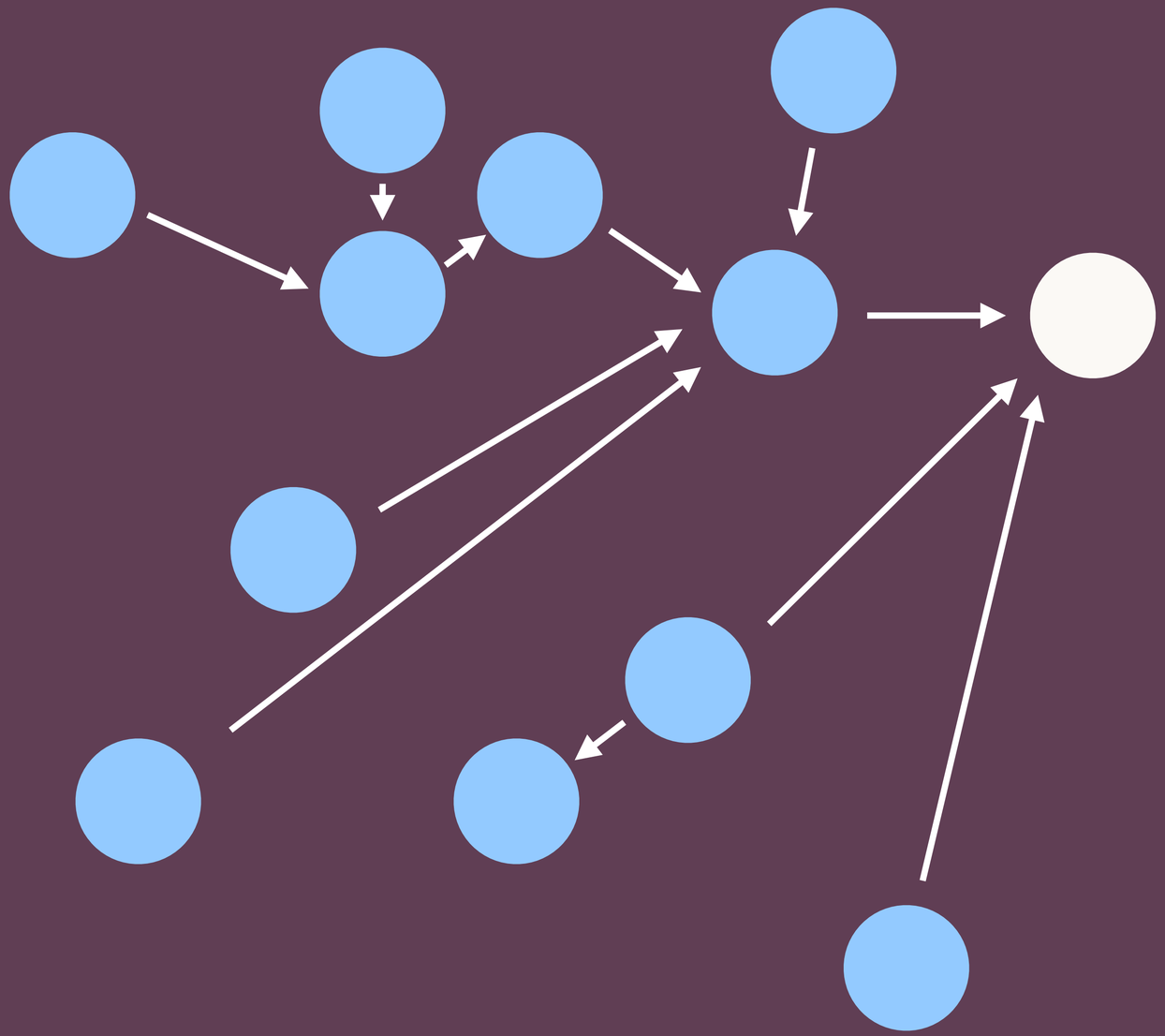
One paper: many possible bits



Each separating line is a possible bit flip.

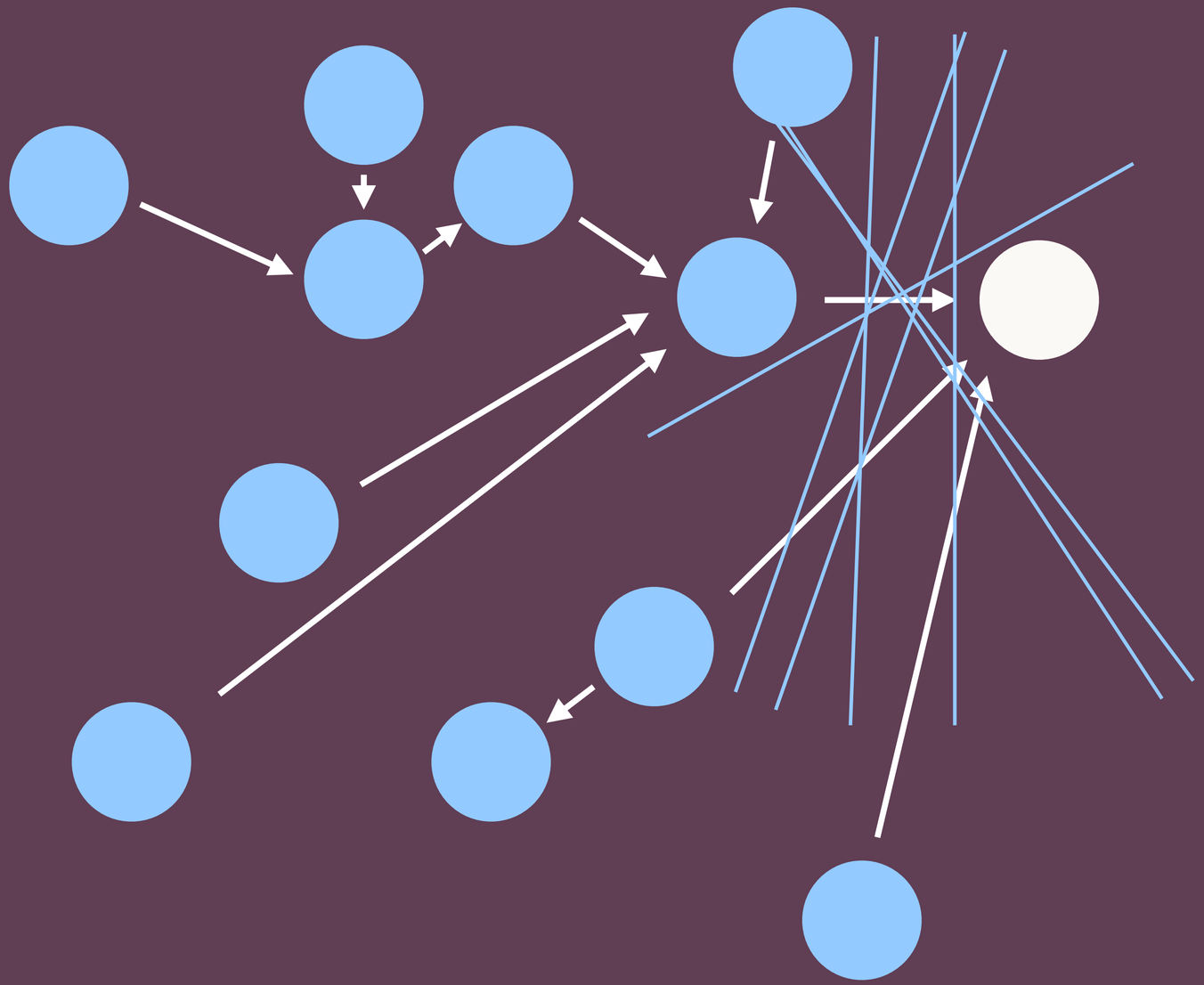
Which one is best?

Literature: clearer bit flip



The broader an understanding you have of the literature and the design axes underneath it, the more effectively you can pick the right bit flip.

Literature: clearer bit flip



The broader an understanding you have of the literature and the design axes underneath it, the more effectively you can pick the right bit flip.

Literature-level bit flip

If your nearest neighbor paper assumed X and you want to do ~X, but other papers in the domain already assumed ~X with slightly different setups, it's a more minor contribution.

Example:

> You are creating a visual question answering algorithm that can look at a picture and answer a question about it. The nearest neighbor used a BERT architecture.
>
> You want to use a modern LLM architecture. That's a paper-level bit flip.
>
> But non-computer vision question answering algorithms already use such an LLM. So it's still a contribution, just not as broad.

Ultimately…

It's unlikely that you will find an idea that nobody has ever articulated in any context ever.

Instead, your goal is to articulate the broadest class of papers possible that your bit flip applies to.

Outcome

This all gets communicated in a Related Work section. (And, to a more limited extent, in the Introduction section.)

A related work section lays out the literature in a way that the reader can understand what you're building on, and what your bit flip is relative to that prior research.

RELATED WORK

In this section, we motivate flash organizations through an integration of the crowdsourcing and organizational design research literature, and connect their design to lessons from distributed work and peer production (Table 1).

Crowdsourcing workflows

Crowdsourcing is the process of making an open call for contributions to a large group of people online [7, 37]. In this paper, we focus especially on *crowd work* [42] (e.g., Amazon Mechanical Turk, Upwork), in which contributors are paid for their efforts. Current crowd work techniques are designed for decomposable tasks that are coordinated by workflows and algorithms [55]. These techniques allow for open-call recruitment at massive scale [67] and have achieved success in modularizable goals such as copyediting [6], real-time transcription [47], and robotics [48]. The workflows can be optimized at runtime among a predefined set of activities [16]. Some even enable collaborative, decentralized coordination instead of step-by-step instructions [46, 86]. As the area advanced, it began to make progress in achieving significantly more complex and interdependent goals [43], such as knowledge aggregation [30], writing [43, 61, 78], ideation [84, 85], clustering [12], and programming [11, 50].

One major challenge to achieving complex goals has been that microtask workflows struggle when the crowd must define new behaviors as work progresses [43, 44]. If crowd workers cannot be given plans in advance, they must form such action plans themselves [51]. However, workers do not always have the context needed to author correct new behaviors [12, 81], resulting in inconsistent or illogical changes that fall short of the intended outcome [44].

Recent work instead sought to achieve complex goals by moving from microtask workers to expert workers. Such systems now support user interface prototyping [70], question-answering and debugging for software engineers [11, 22, 50], worker management [28, 45], remote writing tasks [61], and skill training [77]. For example, flash teams demonstrated that expert workflows can achieve far more complex goals than can be accomplished using microtask workflows [70]. We in fact piloted the current study using the flash teams approach, but the flash teams kept failing at complex and open-ended goals because these goals could not be fully decomposed a priori. We realized that flash teams, like other crowdsourcing approaches, still relied on immutable workflows akin to an assembly line. They always used the same pre-specified sequence of tasks, roles, and dependencies.

Rather than structuring crowds like assembly lines, flash organizations structure crowds like organizations. This perspective implies major design differences from flash teams. First, workers no longer rely on a workflow to know what to do; instead, a centralized hierarchy enables more flexible, de-individuated coordination without pre-specifying all workers' behaviors. Second, flash teams are restricted to fixed tasks, roles, and dependencies, whereas flash organizations introduce a pull request model that enables them to fully reconfigure any organizational structure enabling open-ended adaptation that flash teams cannot achieve. Third, whereas flash teams hire the entire team at once in the beginning, flash organizations' adaptation means the role structure changes throughout the project, requiring on-demand hiring and onboarding. Taken together, these affordances enable flash organizations to scale to much larger sizes than flash teams, and to accomplish more complex and open-ended goals. So, while flash teams' predefined workflows enable automation and optimization, flash organizations enable open-ended adaptation.

Organizational design and distributed work

Flash organizations draw on and extend principles from organizational theory. Organizational design research theorizes how a set of customized organizational structures enable coordination [52]. These structures establish (1) roles that encode the work responsibilities of individual actors [41], (2) groupings of individuals (such as teams) that support local problem-solving and interdependent work [13, 29], and (3) hierarchies that support the aggregation of information and broad communication of centralized decisions [15, 87]. Flash organizations computationally represent these structures, which allows them to be visualized and edited, and uses them to guide work and hire workers. Some organizational designs (e.g., holacracy) are beginning to computationally embed organizational structures, but flash organizations are the first centralized organizations that exist entirely online, with no offline complement. Organizational theory also describes how employees and employers are typically matched through the employee's network [23], taking on average three weeks for an organization to hire [17]. Flash organizations use open-calls to online labor markets to recruit interested workers on-demand, which differs dramatically from traditional organizations and requires different design choices and coordination mechanisms.

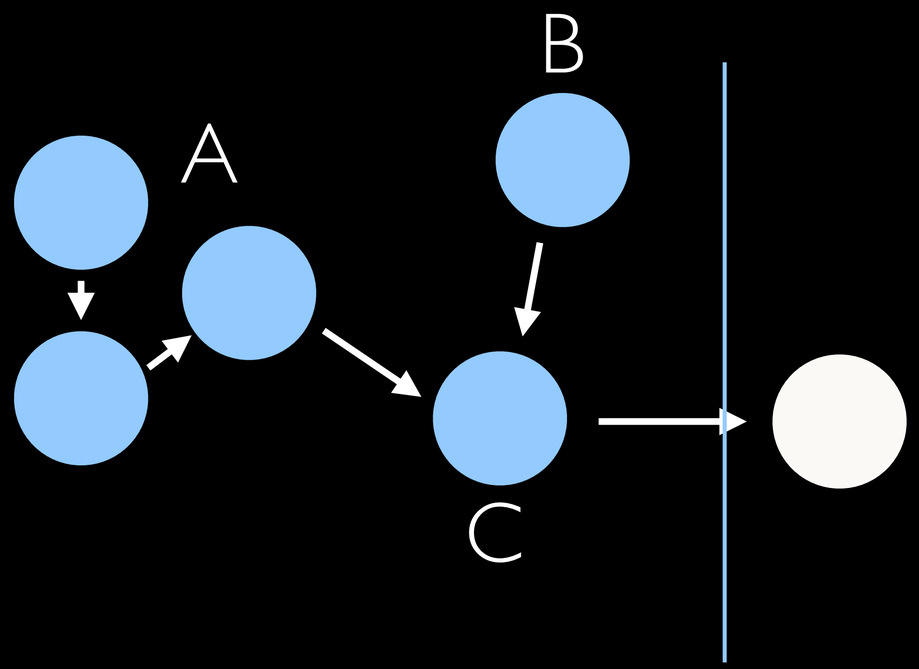
Organizational design research also provides important insight into virtual and distributed teams. Many of the features afforded by collocated work, such as information exchange [64] and shared context [14], are difficult to replicate in distributed and online environments. Challenges arise due to language and cultural barriers [62, 34], incompatible time zones [65, 68], and misaligned incentives [26, 66]. Flash organizations must design for these issues, especially because the workers will not have met before. We designed our system using best practices for virtual coordination, such as loosely coupled work structures [35, 64], situational awareness [20, 27], current state visualization [10, 57], and rich communication tools [64].

Peer production

Flash organizations also relate to peer production [3]. Peer production has produced notable successes in Wikipedia and in free and open source software. One of the main differences between flash organizations and peer production is whether idea conception, decision rights, and task execution are centralized or decentralized. Centralization, for example through a leadership hierarchy, supports tightly integrated work [15, 87]; decentralization, as in wiki software, supports more loosely coupled work. Peer production tends to be decentralized, which offers many benefits, but does not easily support integration across modules [4, 33], limiting the complexity of the resulting work [3]. Flash organizations, in contrast, use centralized structures to achieve integrated planning and coordination,

Outcome

Each part of related work is laying out a subset of the literature, and a bit flip.



A

B

C

RELATED WORK

In this section, we motivate flash organizations through an integration of the crowdsourcing and organizational design research literature, and connect their design to lessons from distributed work and peer production (Table 1).

Crowdsourcing workflows

Crowdsourcing is the process of making an open call for contributions to a large group of people online [7, 37]. In this paper, we focus especially on *crowd work* [42] (e.g., Amazon Mechanical Turk, Upwork), in which contributors are paid for their efforts. Current crowd work techniques are designed for decomposable tasks that are coordinated by workflows and algorithms [55]. These techniques allow for open-call recruitment at massive scale [67] and have achieved success in modularizable goals such as copyediting [6], real-time transcription [47], and robotics [48]. The workflows can be optimized at runtime among a predefined set of activities [16]. Some even enable collaborative, decentralized coordination instead of step-by-step instructions [46, 86]. As the area advanced, it began to make progress in achieving significantly more complex and interdependent goals [43], such as knowledge aggregation [30], writing [43, 61, 78], ideation [84, 85], clustering [12], and programming [11, 50].

One major challenge to achieving complex goals has been that microtask workflows struggle when the crowd must define new behaviors as work progresses [43, 44]. If crowd workers cannot be given plans in advance, they must form such action plans themselves [51]. However, workers do not always have the context needed to author correct new behaviors [12, 81], resulting in inconsistent or illogical changes that fall short of the intended outcome [44].

Recent work instead sought to achieve complex goals by moving from microtask workers to expert workers. Such systems now support user interface prototyping [70], question-answering and debugging for software engineers [11, 22, 50], worker management [28, 45], remote writing tasks [61], and skill training [77]. For example, flash teams demonstrated that expert workflows can achieve far more complex goals than can be accomplished using microtask workflows [70]. We in fact piloted the current study using the flash teams approach, but the flash teams kept failing at complex and open-ended goals because these goals could not be fully decomposed a priori. We realized that flash teams, like other crowdsourcing approaches, still relied on immutable workflows akin to an assembly line. They always used the same pre-specified sequence of tasks, roles, and dependencies.

Rather than structuring crowds like assembly lines, flash organizations structure crowds like organizations. This perspective implies major design differences from flash teams. First, workers no longer rely on a workflow to know what to do; instead, a centralized hierarchy enables more flexible, de-individuated coordination without pre-specifying all workers' behaviors. Second, flash teams are restricted to fixed tasks, roles, and dependencies, whereas flash organizations introduce a pull request model that enables them to fully reconfigure any organizational structure enabling open-ended adaptation that flash teams cannot achieve. Third, whereas flash teams hire the entire team at once in the beginning, flash organizations' adaptation means the role structure changes throughout the project, requiring on-demand hiring and onboarding. Taken together, these affordances enable flash organizations to scale to much larger sizes than flash teams, and to accomplish more complex and open-ended goals. So, while flash teams' predefined workflows enable automation and optimization, flash organizations enable open-ended adaptation.

Organizational design and distributed work

Flash organizations draw on and extend principles from organizational theory. Organizational design research theorizes how a set of customized organizational structures enable coordination [52]. These structures establish (1) roles that encode the work responsibilities of individual actors [41], (2) groupings of individuals (such as teams) that support local problem-solving and interdependent work [13, 29], and (3) hierarchies that support the aggregation of information and broad communication of centralized decisions [15, 87]. Flash organizations computationally represent these structures, which allows them to be visualized and edited, and uses them to guide work and hire workers. Some organizational designs (e.g., holacracy) are beginning to computationally embed organizational structures, but flash organizations are the first centralized organizations that exist entirely online, with no offline complement. Organizational theory also describes how employees and employers are typically matched through the employee's network [23], taking on average three weeks for an organization to hire [17]. Flash organizations use open-calls to online labor markets to recruit interested workers on-demand, which differs dramatically from traditional organizations and requires different design choices and coordination mechanisms.

Organizational design research also provides important insight into virtual and distributed teams. Many of the features afforded by collocated work, such as information exchange [64] and shared context [14], are difficult to replicate in distributed and online environments. Challenges arise due to language and cultural barriers [62, 34], incompatible time zones [65, 68], and misaligned incentives [26, 66]. Flash organizations must design for these issues, especially because the workers will not have met before. We designed our system using best practices for virtual coordination, such as loosely coupled work structures [35, 64], situational awareness [20, 27], current state visualization [10, 57], and rich communication tools [64].

Peer production

Flash organizations also relate to peer production [3]. Peer production has produced notable successes in Wikipedia and in free and open source software. One of the main differences between flash organizations and peer production is whether idea conception, decision rights, and task execution are centralized or decentralized. Centralization, for example through a leadership hierarchy, supports tightly integrated work [15, 87]; decentralization, as in wiki software, supports more loosely coupled work. Peer production tends to be decentralized, which offers many benefits, but does not easily support integration across modules [4, 33], limiting the complexity of the resulting work [3]. Flash organizations, in contrast, use centralized structures to achieve integrated planning and coordination,

Performing a literature search

“An hour in the library saves you a year at the keyboard.”

Goal: build the literature graph

While I don't literally draw out the graph, building up that understanding in my head and visualizing a few different axes is key in identifying the right bit flip.

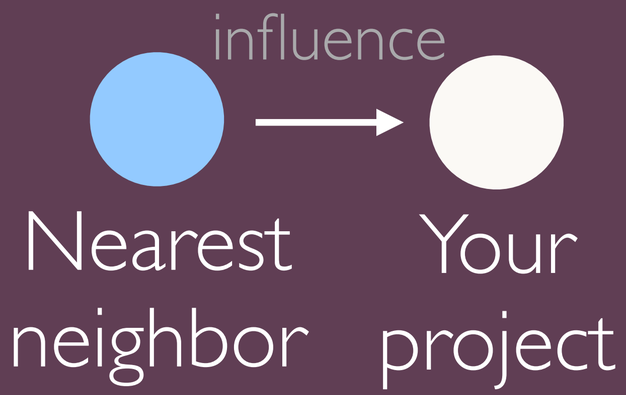
So how do we get there?

Step one: nearest neighbor

Start with a seed paper that is the closest in the design space to yours. This is your **nearest neighbor** paper.

Read this paper in depth. Understand it.

(Your nearest neighbor paper may not be a great paper! Often great ideas are adjacent to a near miss.)


influence
Nearest neighbor
Your project

Step two: expand your horizon

What are the 3–5 most important citations to that nearest neighbor?

- Look at the papers that it cited visibly and carefully in the Introduction and Related Work
- Look at which papers it is arguing a bit flip from

Are those most important citations staying in the neighborhood of your topic, or going somewhere else? Keep the ones that are staying in the neighborhood.

How to expand the horizon

Backward influence: influential citations in the papers that you've read

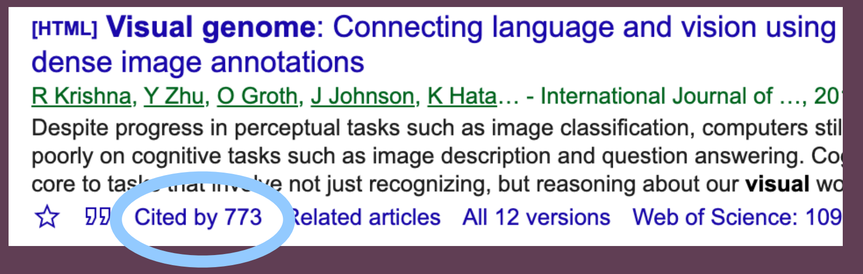
Tools: reading

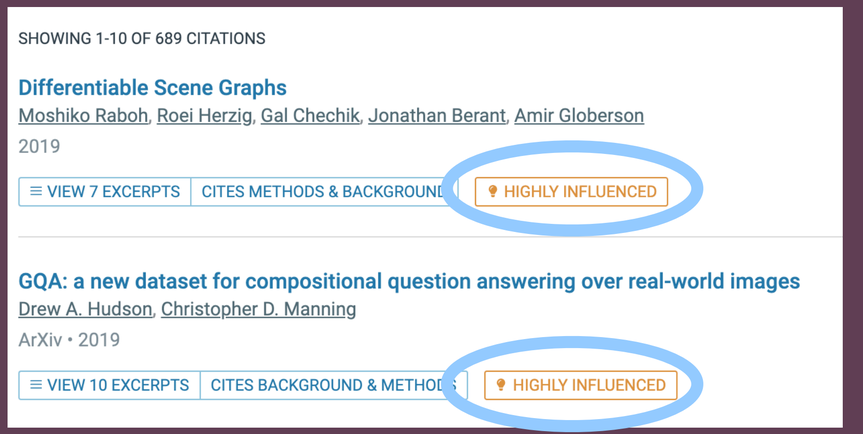
Forward influence: papers citing the ones that you've read

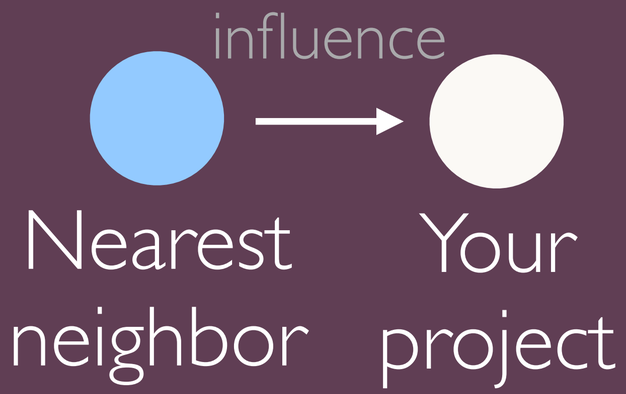
Tools: Google Scholar's "Cited By", Semantic Scholar's "highly influenced" designation

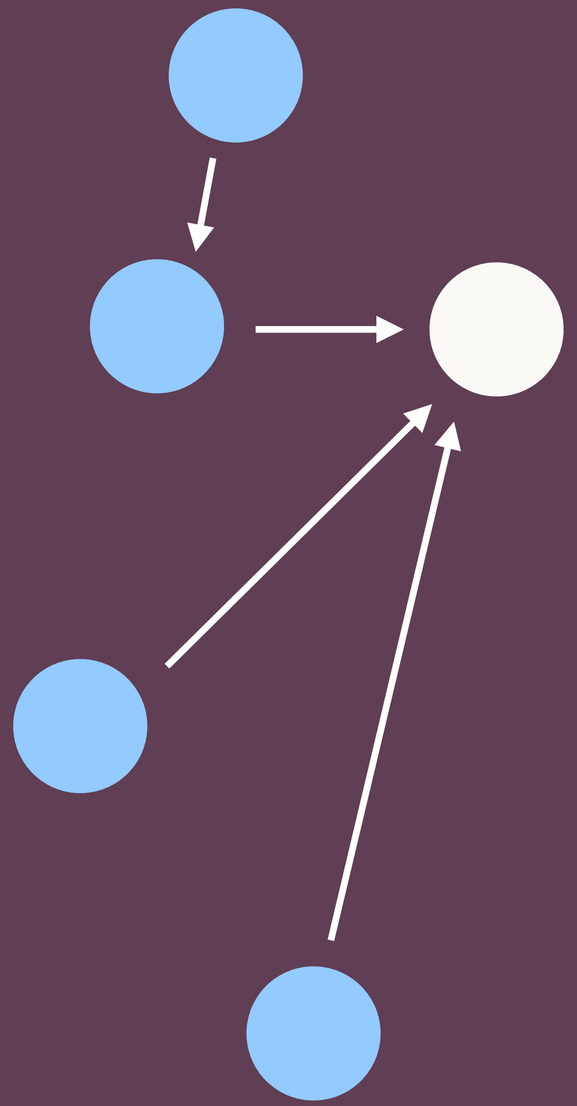
Relatedness: contemporaneous but not citing

Tools: Google Scholar's "Related articles"

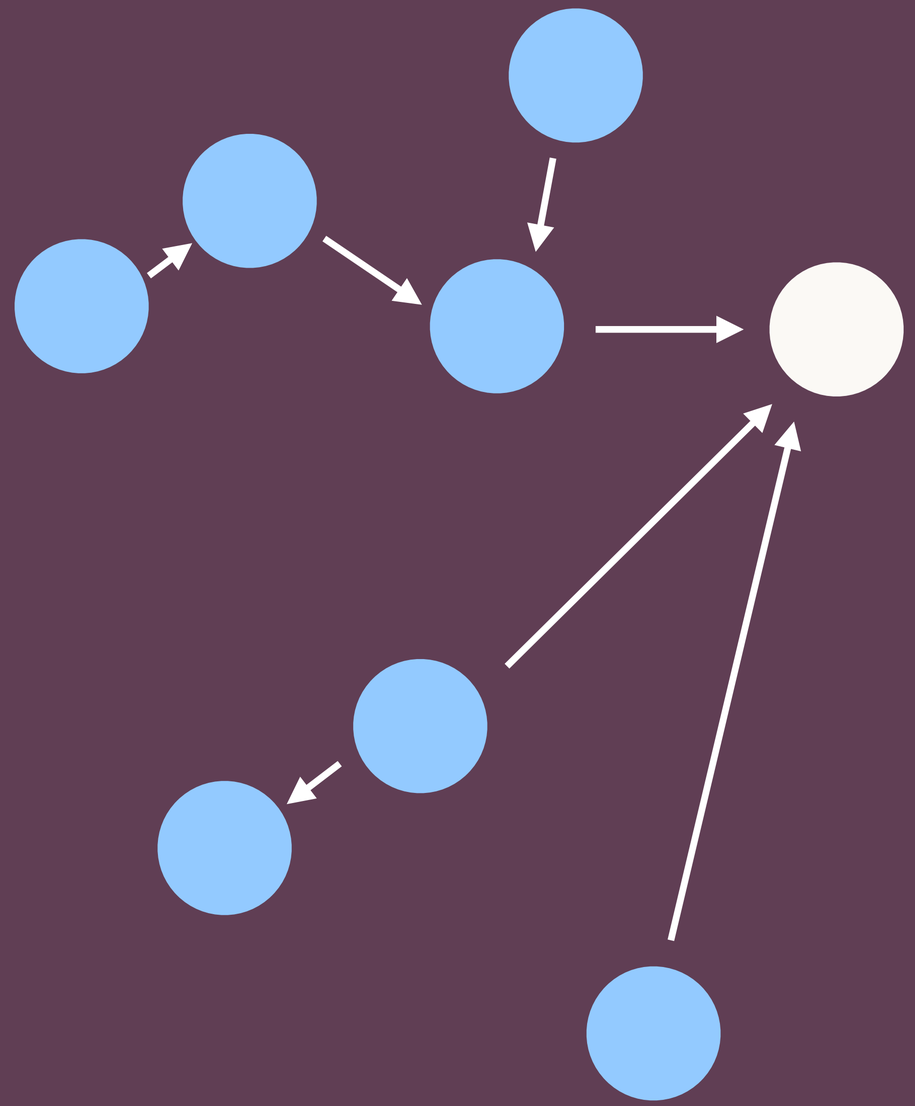




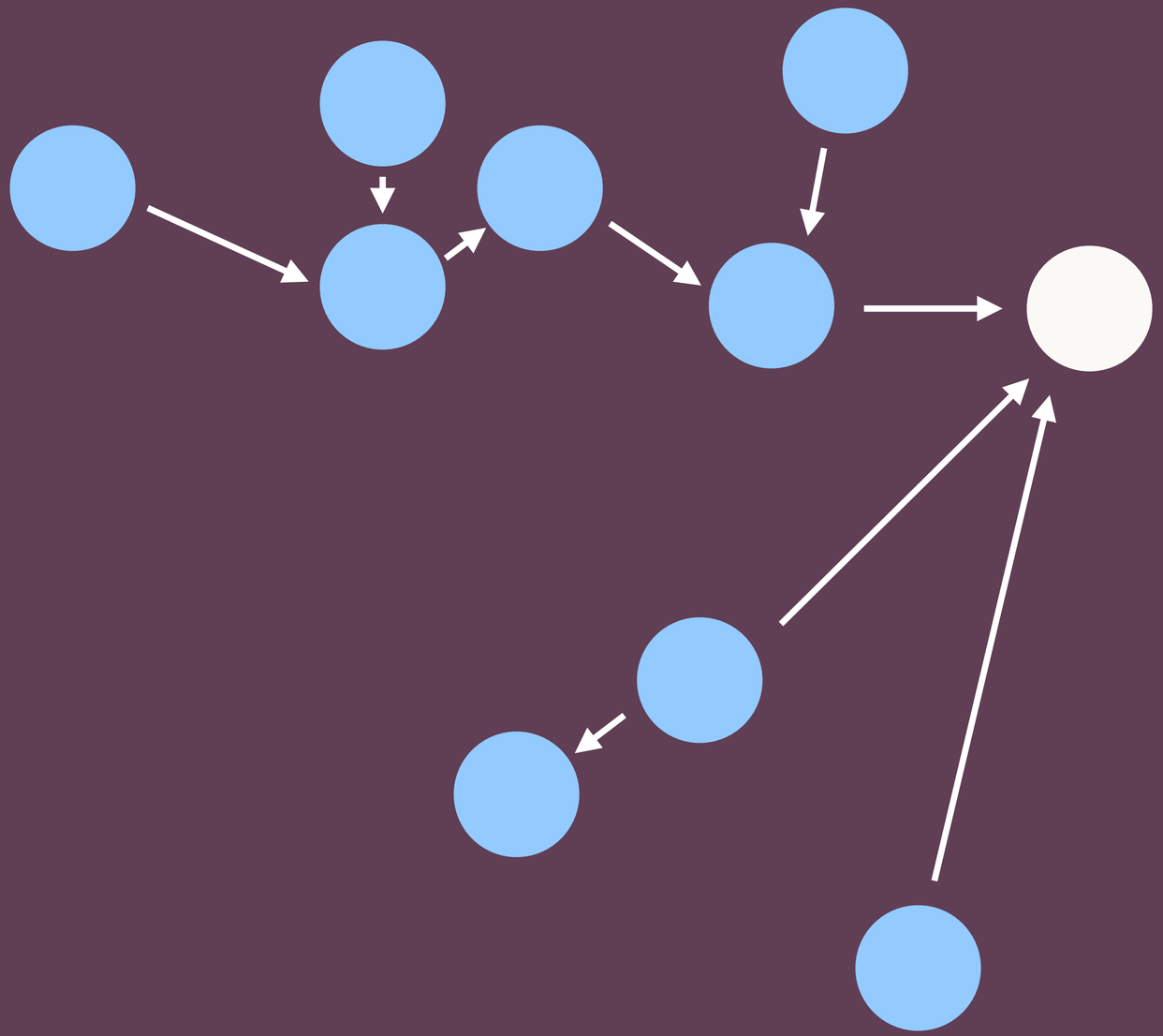

influence
Nearest neighbor
Your project



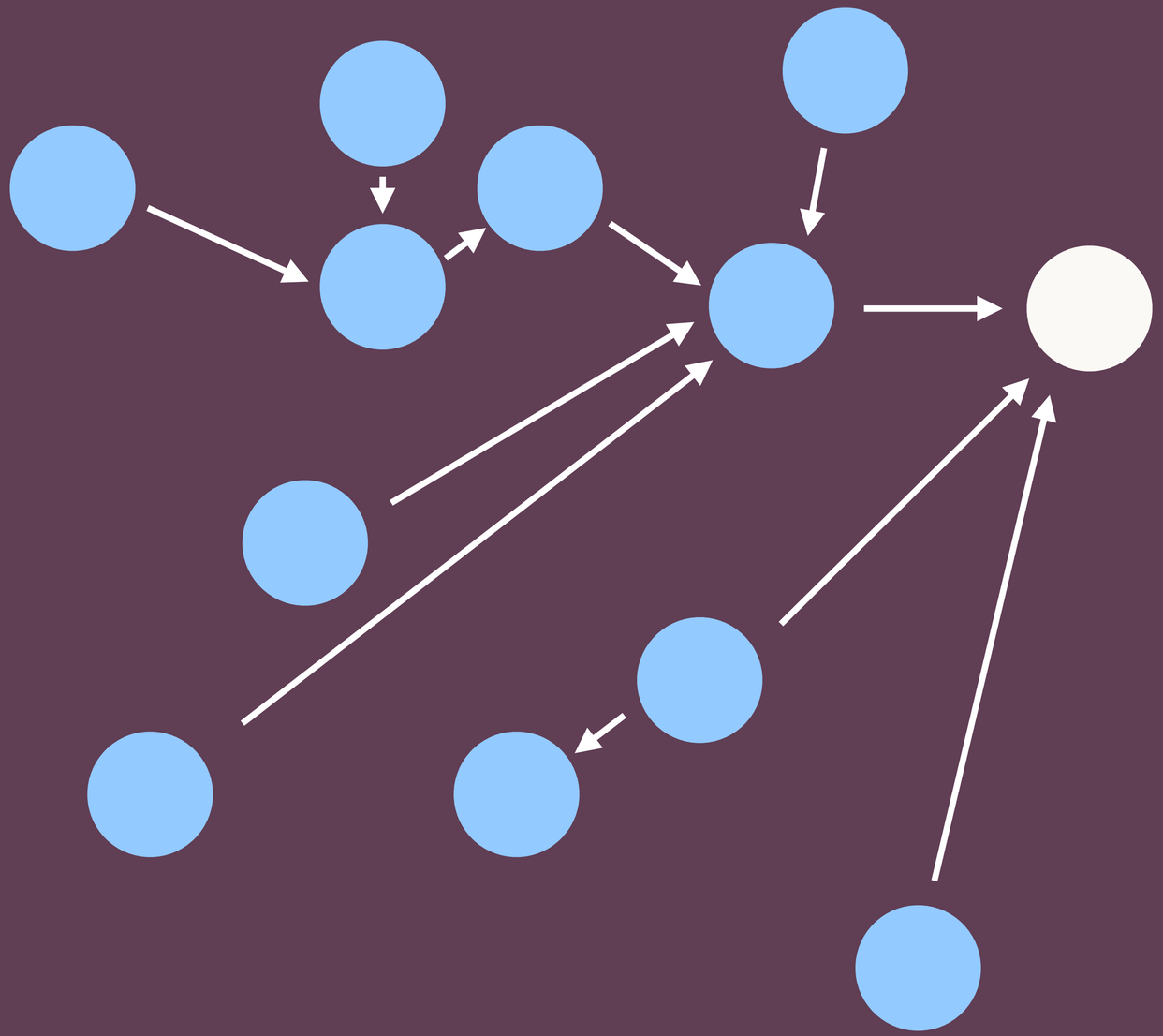
First expansion



Second expansion



continuing…



Final set

Filtering your horizon

Not all papers achieve the same level of quality. Especially on white paper archives such as arXiv.org, quality can be variable.

How do I know what to read and what to ignore?

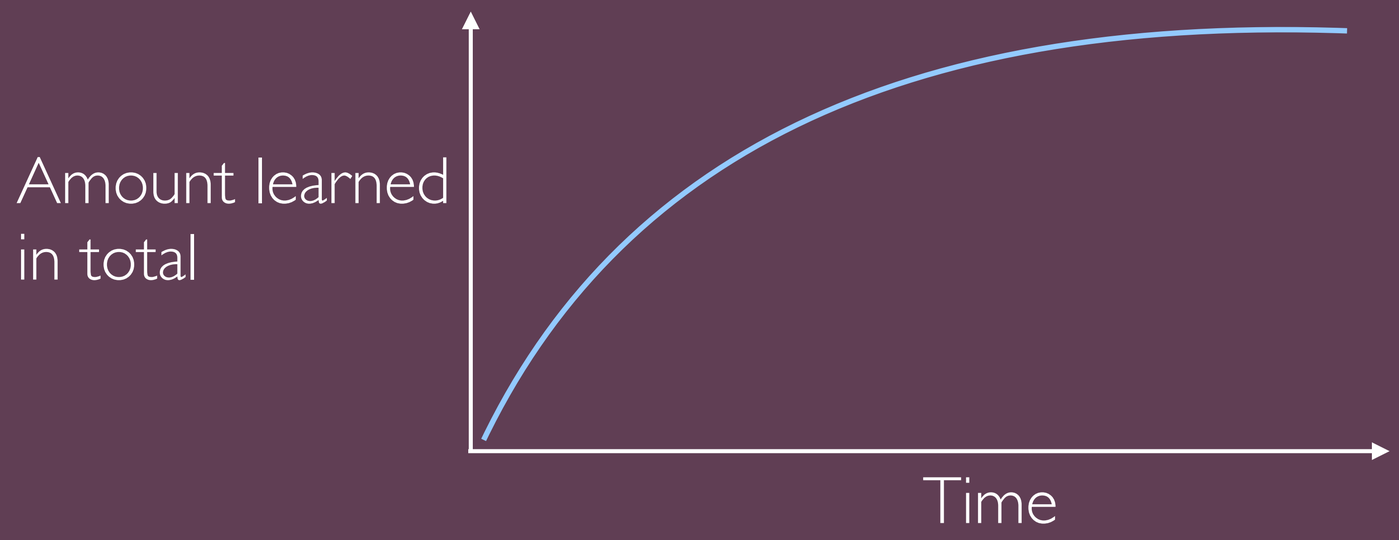
- If the paper is from a reputable venue: ask your TA for the reputable venues in the field of your project, and stick to those venues. (Or ask the TA if you find a relevant paper outside of those venues and want a gut check.)
- Or, if the paper is from a reputable institution or author: again, you might ask your TA whether an institution you've never heard of is reputable

Asymptoting

Keep track of **how much you're learning about the design axes** as you consume the additional papers. Typically, you are learning the most at the very beginning, and the amount per paper starts going down after five papers or so.

A PhD student often asymptotes after 25–35 papers.

For this class, we'll go with 15 for now.



Reading a paper for a literature search

Temptation: understand everything.

Typically, when we come to a paper, we want to understand everything about it. We stop and reread any point we don't get.

This can take an hour or two per paper for a new researcher.

This strategy can be useful at the beginning, but it is actively harmful in constructing a related work section.

Understand the main point

Instead, articulate to yourself: **what is the main point** that this paper is making?

Then, focus your reading and effort most closely on the parts of the paper that are supporting or evaluating that point.

It's OK not to understand every sentence in the paper.

Your goal isn't to understand the paper — it's to understand the literature, what works and what doesn't (and why!), and the bits that are available to flip.

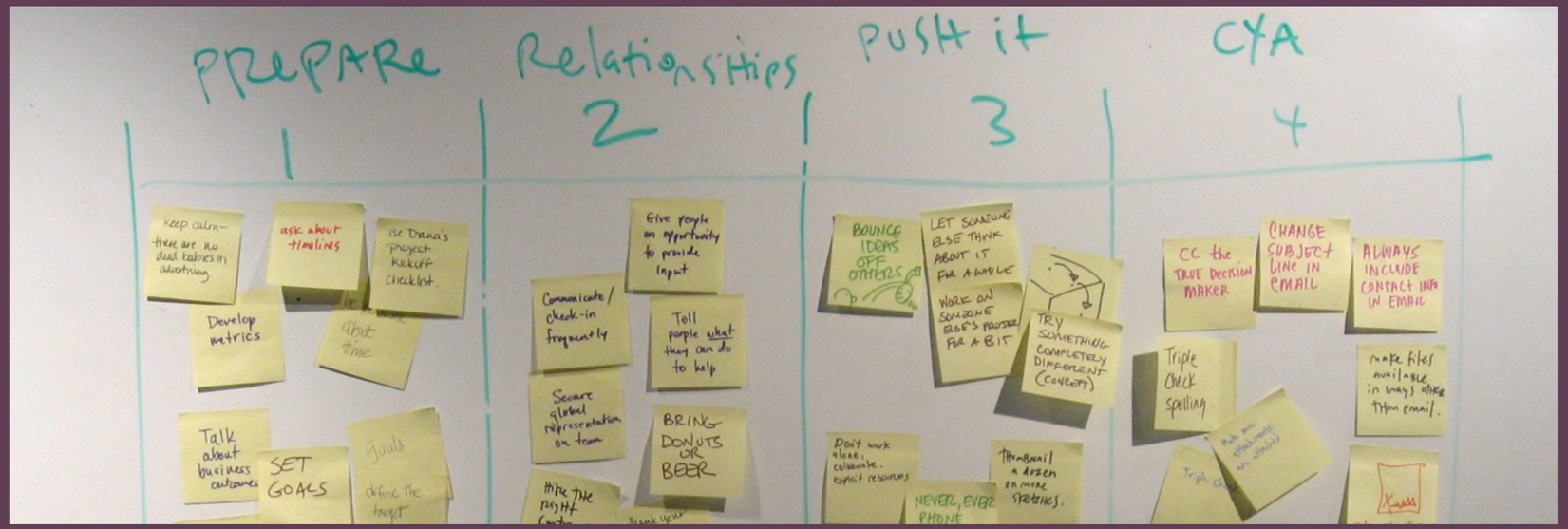
Writing the Related Work

Step 1: affinity mapping

Put each paper onto a post-it note

Place the post-it notes onto a whiteboard or wall, placing similar ideas close to each other.

Group by whatever makes sense to you.



Step 2: regrouping

Typically, these first groups represent topics (nouns). This isn't wrong, but it's not the most helpful in writing a Related Work section.

So, instead, aim for each group to have a shared thesis behind it, not just a noun. Regroup your post-its around shared theses.

Not good:

Autoencoders

Better:

Autoencoding ensures the language model retains the question that it is answering

Step 3: outlining

Each thesis then becomes the topic sentence of a paragraph of the Related Work section

Autoencoding ensures the language model retains the question that it is answering

Visual Question Answering as a task benefits from having question localization

Transformer-based language models dramatically improve performance on the Visual Question Answering task

Step 4: writing

The temptation when writing is to list all the related work under the topic. Don't do this.

Projects have used runtime feedback to help programmers debug projects. The Arsinine project highlighted variables as they are being accessed [4]. Achilles produced a summary report on the command line after execution completed [7]. More recently, Arbuckle played annoying sounds from the computer

Instead, start with the thesis, and use the paragraph to prove the thesis.

Runtime feedback of memory accesses gives programmers an intuitive sense of whether the program's behavior match their intuitions. Highlighting variables as they are accessed provides a rough understanding of the most-used items [4]. Complementing this report with a summary afterwards helps support this sensemaking [7]. Particularly worrisome program behaviors are more likely to be noticed when

Step 4: writing

Once you've summarized the work, articulate the bit flip.

Runtime feedback of memory accesses gives programmers an intuitive sense of whether the program's behaviors match their intuitions. Highlighting variables as they are accessed provides a rough understanding of the most-used items [4]. Complementing this report with a summary afterwards helps support this sensemaking [7]. Particularly worrisome program behaviors are more likely to be noticed

Where this prior work suggests that programmers make fewer errors when given runtime feedback of memory accesses, this project demonstrates that runtime feedback of function call graphs help reduce errors as well. We draw on techniques from the prior literature such as highlighting and post-hoc summaries, and extend these with a visualization algorithm for function calls.

Outcome

Paragraphs with topic sentences, then the bit flip re-articulated and expanded upon

About 600–700 words

RELATED WORK

In this section, we motivate flash organizations through an integration of the crowdsourcing and organizational design research literature, and connect their design to lessons from distributed work and peer production (Table 1).

Crowdsourcing workflows

Crowdsourcing is the process of making an open call for contributions to a large group of people online [7, 37]. In this paper, we focus especially on *crowd work* [42] (e.g., Amazon Mechanical Turk, Upwork), in which contributors are paid for their efforts. Current crowd work techniques are designed for decomposable tasks that are coordinated by workflows and algorithms [55]. These techniques allow for open-call recruitment at massive scale [67] and have achieved success in modularizable goals such as copyediting [6], real-time transcription [47], and robotics [48]. The workflows can be optimized at runtime among a predefined set of activities [16]. Some even enable collaborative, decentralized coordination instead of step-by-step instructions [46, 86]. As the area advanced, it began to make progress in achieving significantly more complex and interdependent goals [43], such as knowledge aggregation [30], writing [43, 61, 78], ideation [84, 85], clustering [12], and programming [11, 50].

One major challenge to achieving complex goals has been that microtask workflows struggle when the crowd must define new behaviors as work progresses [43, 44]. If crowd workers cannot be given plans in advance, they must form such action plans themselves [51]. However, workers do not always have the context needed to author correct new behaviors [12, 81], resulting in inconsistent or illogical changes that fall short of the intended outcome [44].

Recent work instead sought to achieve complex goals by moving from microtask workers to expert workers. Such systems now support user interface prototyping [70], question-answering and debugging for software engineers [11, 22, 50], worker management [28, 45], remote writing tasks [61], and skill training [77]. For example, flash teams demonstrated that expert workflows can achieve far more complex goals than can be accomplished using microtask workflows [70]. We in fact piloted the current study using the flash teams approach, but the flash teams kept failing at complex and open-ended goals because these goals could not be fully decomposed a priori. We realized that flash teams, like other crowdsourcing approaches, still relied on immutable workflows akin to an assembly line. They always used the same pre-specified sequence of tasks, roles, and dependencies.

Rather than structuring crowds like assembly lines, flash organizations structure crowds like organizations. This perspective implies major design differences from flash teams. First, workers no longer rely on a workflow to know what to do; instead, a centralized hierarchy enables more flexible, de-individuated coordination without pre-specifying all workers' behaviors. Second, flash teams are restricted to fixed tasks, roles, and dependencies, whereas flash organizations introduce a pull request model that enables them to fully reconfigure any organizational structure enabling open-ended adaptation that flash teams cannot achieve. Third, whereas flash teams hire the entire team at once in the beginning, flash organizations' adaptation means the role structure changes throughout the project, requiring on-demand hiring and onboarding. Taken together, these affordances enable flash organizations to scale to much larger sizes than flash teams, and to accomplish more complex and open-ended goals. So, while flash teams' predefined workflows enable automation and optimization, flash organizations enable open-ended adaptation.

Organizational design and distributed work

Flash organizations draw on and extend principles from organizational theory. Organizational design research theorizes how a set of customized organizational structures enable coordination [52]. These structures establish (1) roles that encode the work responsibilities of individual actors [41], (2) groupings of individuals (such as teams) that support local problem-solving and interdependent work [13, 29], and (3) hierarchies that support the aggregation of information and broad communication of centralized decisions [15, 87]. Flash organizations computationally represent these structures, which allows them to be visualized and edited, and uses them to guide work and hire workers. Some organizational designs (e.g., holacracy) are beginning to computationally embed organizational structures, but flash organizations are the first centralized organizations that exist entirely online, with no offline complement. Organizational theory also describes how employees and employers are typically matched through the employee's network [23], taking on average three weeks for an organization to hire [17]. Flash organizations use open-calls to online labor markets to recruit interested workers on-demand, which differs dramatically from traditional organizations and requires different design choices and coordination mechanisms.

Organizational design research also provides important insight into virtual and distributed teams. Many of the features afforded by collocated work, such as information exchange [64] and shared context [14], are difficult to replicate in distributed and online environments. Challenges arise due to language and cultural barriers [62, 34], incompatible time zones [65, 68], and misaligned incentives [26, 66]. Flash organizations must design for these issues, especially because the workers will not have met before. We designed our system using best practices for virtual coordination, such as loosely coupled work structures [35, 64], situational awareness [20, 27], current state visualization [10, 57], and rich communication tools [64].

Peer production

Flash organizations also relate to peer production [3]. Peer production has produced notable successes in Wikipedia and in free and open source software. One of the main differences between flash organizations and peer production is whether idea conception, decision rights, and task execution are centralized or decentralized. Centralization, for example through a leadership hierarchy, supports tightly integrated work [15, 87]; decentralization, as in wiki software, supports more loosely coupled work. Peer production tends to be decentralized, which offers many benefits, but does not easily support integration across modules [4, 33], limiting the complexity of the resulting work [3]. Flash organizations, in contrast, use centralized structures to achieve integrated planning and coordination,

Progress Report 1

At this point, your project transitions to a state where your team is starting to take steps to execute the project.

For the first week, your TA should have given you a goal for your first steps. Each week through the rest of the quarter, your team will submit a brief summary:

- This week's plan
- This week's result
- Next week's plan
- Due: the night before section, on Canvas. Details at cs197.stanford.edu.

Assignment 2

Perform a literature search for your project, alongside your group

- Read the nearest neighbor paper, provided by your TA
- Expand your literature search to ~15 papers
- Affinity diagramming
- Articulating a bit flip
- Writing a Related Work section for your final paper

Due: next Thursday on Canvas

Details at cs197.stanford.edu

Computer Science Research