Proposed Solutions

Typically, after the introductory material, we explain the proposed solutions (usually some kind of software, algorithm, protocol, etc.). How this is written depends almost entirely on the content.

After reading this, the reader should be able to answer:

- **What do we do?** This is the mechanics of what's done. It should be **new**. It should be **reproducible**.
- **How does it achieve our goal?** How does What do we do? solve our problem?
- **Why choose this method?** There are other ways to solve the same problem—why do we choose this way to achieve our goal?

This is the bare minimum.

Other things we might do:

- We might also split into subgoals, and answer the above for each subgoal.
- We might describe critical previous work in detail (which needs to be clearly identified).
- We might need to give intermediate experimental results (e.g. profiling) which helps answer Why choose this method?.

What do we do?

After the minimal (worded) description in the Introduction (or thereabouts)...

- Next, we usually **describe the hardware, software, or mathematics** in detail. (Might be part of the introductory material.)
  - We give **only what is needed to understand the paper**. (We give references for further reading.)
  - *Structure*: This should be **skippable** (usually achieved by using sections or subsections). It's boring, but necessary to write.
- Next, we usually give **increasingly specific explanations** of what we do using what we just introduced. *Structure*: I recommend **at least three layers**.

There's no formula or template here: **every paper is optimized differently**. Try several ways, and use the best:

> *... you must organize what you want to say, and you must arrange it in the order you want it said in, you must* **write it, rewrite it, and re-rewrite it** *several times, ...*
>
> — *Halmos, How to write mathematics, 2009.*

A typical example

Jalbert and Sen (2010), *A Trace Simplification Technique for Effective Debugging of Concurrent Programs*, FSE-18, 2010:

- Section 1: Introduction.
  - (What do we do? [1-st layer]) "The contributions of this paper ...".
- Section 2: Overview.
  - (What do we do? [2-nd layer]) Description of the proposed algorithm.
  - (How does it achieve our goal?) Illustration of how it functions using a **toy example**.
- Section 3: Algorithm.
  - Precise description of the solution.
    1. Local definitions,
    2. (Why choose this method?) theoretical analysis, and
    3. (What do we do? [3-rd layer]) **pseudocode**.
- Section 4: Implementation.
  - (What do we do? [4-th layer]) How they choose to implement the algorithm.
- Sections 5–9: ...

This is about as typical as it gets; it varies greatly.

Using a **increasingly detailed layered structure** allows the reader to decide on their level of commitment to the paper:

1. Simple technical description of the key elements.
   - Perhaps this is all the reader needs. (E.g., all an expert would need; or the reader might want just want a rough idea.)
   - The reader might not be able to (easily) understand the full details.
2. Descriptions of each key element.
   - Facilitated by aids e.g., flow charts or toy examples. Think: **what would I write on a whiteboard** to explain this to someone else?
   - Probably the reader will look at the aids, and mentally reverse-engineer the main text.
3. Full details.
   - Facilitated by pseudocode, well-chosen notation and terminology, well-presented equations, correct typesetting, etc.
   - Whatever a graduate student would need to reproduce the findings.
   - High risk of this being boring!! It needs structure!

All of these improve with LaTeX skills.

Toy examples

Toy examples

A toy example is an unrealistically simple, made-up example we use to illustrate a point. We use them to **explain some underlying property** to the reader. We get rid of everything else.

They're useful for answering: How does what we do achieve our goal?

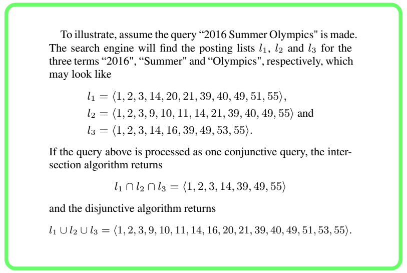
Two important forms are:

- *A running example.* Alongside a step-to-step description of the technicalities, we explain what it means for an example.
  These are useful e.g. for when there's lots of mathematics. The reader can check they understand the mathematics correctly on the running example.
- *A figure.* These are useful e.g. for the following:
  - To illustrate an algorithm's underlying mechanism. I.e., how it achieves what it achieves.
  - To illustrate an algorithm's structure or workflow.

Running example

Zhang et al., SIGIR, 2016 (includes me as co-author):

Page 2



To illustrate, assume the query “2016 Summer Olympics" is made. The search engine will find the posting lists l_1, l_2 and l_3 for the three terms “2016", “Summer" and “Olympics", respectively, which may look like

$$l_1 = \langle 1, 2, 3, 14, 20, 21, 39, 40, 49, 51, 55 \rangle,$$
$$l_2 = \langle 1, 2, 3, 9, 10, 11, 14, 21, 39, 40, 49, 55 \rangle \text{ and}$$
$$l_3 = \langle 1, 2, 3, 14, 16, 39, 49, 53, 55 \rangle.$$

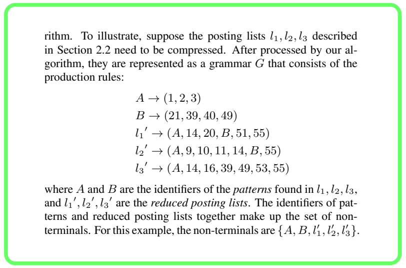
If the query above is processed as one conjunctive query, the intersection algorithm returns

$$l_1 \cap l_2 \cap l_3 = \langle 1, 2, 3, 14, 39, 49, 55 \rangle$$

and the disjunctive algorithm returns

$$l_1 \cup l_2 \cup l_3 = \langle 1, 2, 3, 9, 10, 11, 14, 16, 20, 21, 39, 40, 49, 51, 53, 55 \rangle.$$

Page 3



rithm. To illustrate, suppose the posting lists l_1, l_2, l_3 described in Section 2.2 need to be compressed. After processed by our algorithm, they are represented as a grammar G that consists of the production rules:

$$A \to (1, 2, 3)$$
$$B \to (21, 39, 40, 49)$$
$${l_1}' \to (A, 14, 20, B, 51, 55)$$
$${l_2}' \to (A, 9, 10, 11, 14, B, 55)$$
$${l_3}' \to (A, 14, 16, 39, 49, 53, 55)$$

where A and B are the identifiers of the *patterns* found in l_1, l_2, l_3, and ${l_1}', {l_2}', {l_3}'$ are the *reduced posting lists*. The identifiers of patterns and reduced posting lists together make up the set of non-terminals. For this example, the non-terminals are $\{A, B, l'_1, l'_2, l'_3\}$.

The running example is used again on pages 5–6.

As we introduce the formalities, we also explain its relevance to this running example.

(There's some grammar problems and tense issues here.)

Toy-example figures

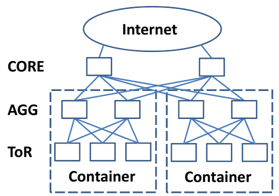
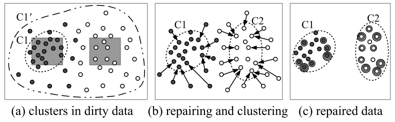


Figure 1: An example scale-out DCN topology.

— Wu et al., SIGCOMM, 2012.



(a) clusters in dirty data (b) repairing and clustering (c) repaired data

Figure 1: Clustering with repairing over dirty data

— Song et al., KDD, 2015.

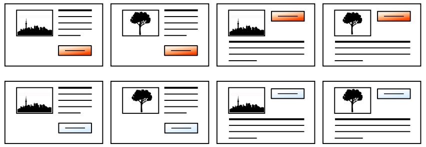


Figure 3: Example of a multivariate test, in which three sections (image, button colour, text placement) are tested, each one with two variations.

— Holzmann and Hutflesz, MoMM, 2014.

What are the pros and cons of this toy example?

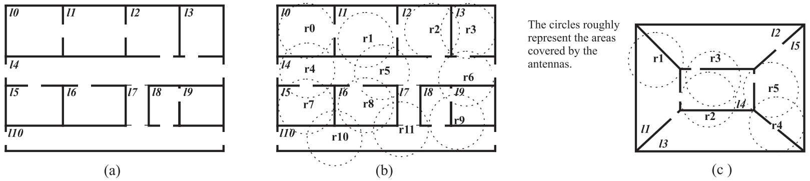

l0
l1
l2
l3
l4
l5
l6
l7
l8
l9
l10
r0
r1
r2
r3
r4
r5
r6
r7
r8
r9
r10
r11
The circles roughly represent the areas covered by the antennas.
(a)
(b)
(c)

Fig. 1. (*a*) A floor of a building; (*b*) positions of the readers in the floor; (*c*) a map used in the examples.

— Fazzinga et al., ACM Trans. Database Syst., 2016.

- **The math is not in math mode!**
- The text is part of the image (**not searchable**). The **font is too small**. (Should be part of the caption.)
- There's an unwanted space in "(c)". Sloppiness.
- The two (a)'s are different. Sloppiness.
- One subfigure starts with "r0" and "l0", another starts with "r1" and "l1". Inconsistent.
- We don't need to repeat the room labels in (b). Clutter.

Visualizations

A wide range of informations types can be depicted in a wide variety of ways. These **visualizations make information easier to absorb and understand**.

We generally use this structure:

- **A figure.** We draw the information in some way. We minimize words and notation. We try to be consistent.
- **A figure caption.** Ideally, a figure together with its caption is self-contained.
- **A detailed description in the main text.**

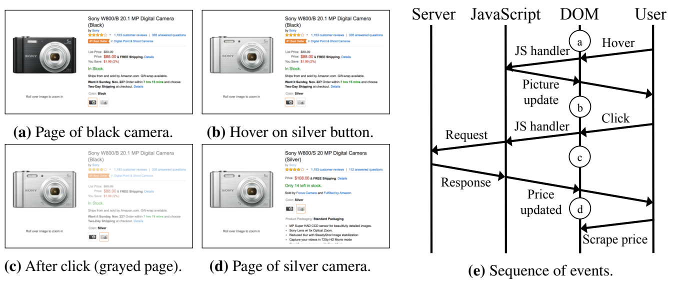
We **don't want to oversimplify**, but we need to **resist the urge to include everything**: only the key points.

Figures are highly visible; they need to be **worthy of publication**.

Low-quality figures implies the authors have not spent enough time on the paper; this implies the paper will likely contain low-quality results.

An excellent visualization

This visualization deficits a **timeline** of a hypothetical user's behaviors:



(a) Page of black camera. **(b)** Hover on silver button. **(c)** After click (grayed page). **(d)** Page of silver camera. **(e)** Sequence of events.

Figure 1: Amazon price scraping interaction. Each circle in (e) corresponds to a webpage state shown in (a)-(d). Note that hovering over the silver option instantly displays the silver camera picture but not its price. Only after the click does the page request the silver price from the server and overlays to gray. The response updates the price and removes the gray overlay.

— Barman et al., OOPSLA, 2016.

Lots of information! Succinct; no clutter; exactly what is needed.

A decent visualization

This visualization plots the **developmental relationships** between prior work:

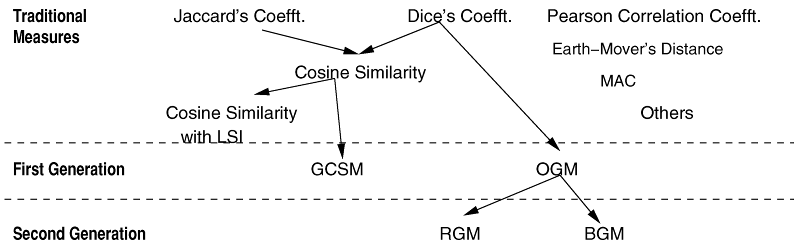


Fig. 2. Evolution of similarity measures.

— Ganesan et al., ACM Trans. Inf. Syst., 2003.

It's helpful, but it's not presented perfectly:

- Font does not match text.
- Arrows and lines obstruct text; poor alignment.
- Text is part of the image. Not scalable; not searchable.

A messy visualization

This visualization plots a **hierarchy** in music:

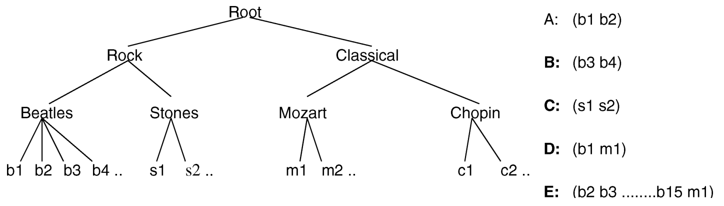
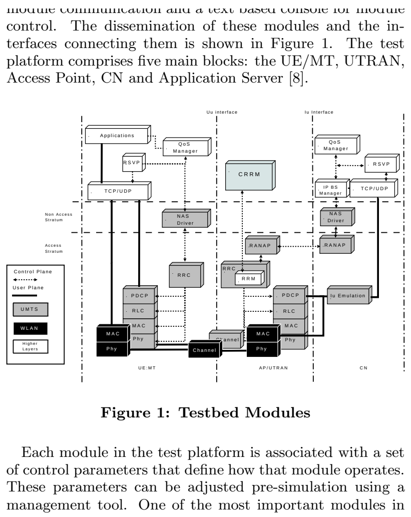


Fig. 1. Music CD hierarchy.

— Ganesan et al., ACM Trans. Inf. Syst., 2003.

- Font does not match text; font in "s1" and "s2" different.
- Inconsistent ellipses (...).
- Lines obstruct text; asymmetry.
- Text is part of the image. Not scalable; not searchable.
- **The math is not in math mode!**
  (It should be b_1, b_2, etc., not b1, b2, etc.)

A poor visualization



module communication and a text based console for module control. The dissemination of these modules and the interfaces connecting them is shown in Figure 1. The test platform comprises five main blocks: the UE/MT, UTRAN, Access Point, CN and Application Server [8].

Figure 1: Testbed Modules

Each module in the test platform is associated with a set of control parameters that define how that module operates. These parameters can be adjusted pre-simulation using a management tool. One of the most important modules in

— Skehill et al., Tridentcom, 2008.

This visualization plots some kind of **flow chart**.

The **font is too small!** (It should be the same size as the main text.)

Too much crammed in.

What's wrong with this visualization?

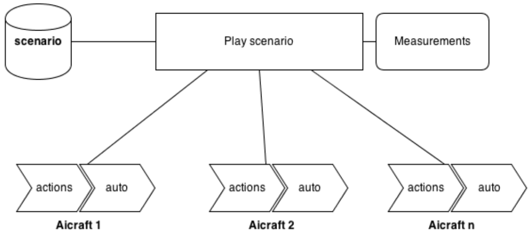


Figure 2 - Core Simulator

— Frau, ATACCS, 2013.

- **Overly simple.**
- The text is not scalable nor searchable.
- The font doesn't match the font of the paper.
- The math is not in math mode.
- "Aircraft" is spelled wrong: **no spell-checker in figures**.
- Inconsistent capitalization.

What are the pros and cons of this visualization?

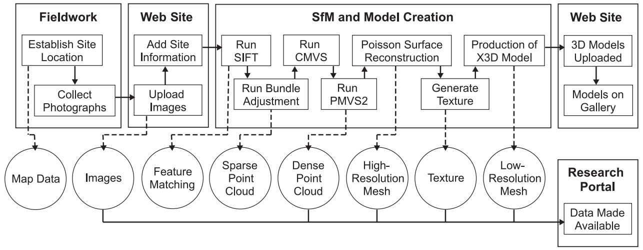


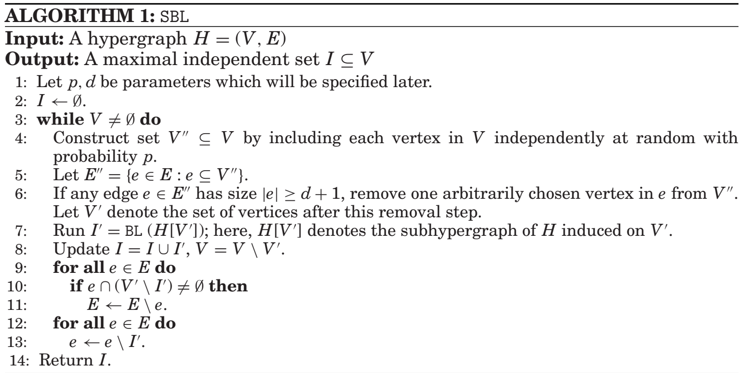
Fig. 1. The work flow of HeritageTogether, from the initial photography fieldwork to the model creation and distribution via the Web site and research portal. Processes are contained within rectangles, and data produced from the processes are contained within circles; data is produced from the processes marked with the dashed lines.

— Miles et al., ACM J. Comp. Cult. Heritage, 2015.

- *Pro*: Not overly simple. Not overly complicated.
- *Pro*: Structured (!).
- *Pro*: Consistent (!).
- *Con*: The text is part of the image (font doesn't match; text is not searchable).

Pseudocode

A typical example

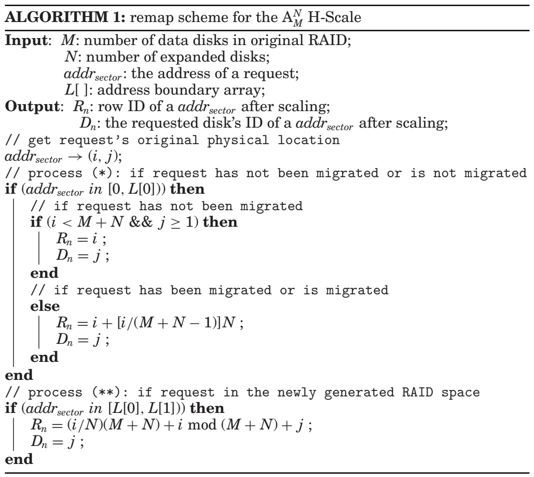


```
ALGORITHM 1: SBL
Input: A hypergraph $H = (V, E)$
Output: A maximal independent set $I \subseteq V$
 1: Let $p, d$ be parameters which will be specified later.
 2: $I \leftarrow \emptyset$.
 3: while $V \neq \emptyset$ do
 4:    Construct set $V'' \subseteq V$ by including each vertex in $V$ independently at random with probability $p$.
 5:    Let $E'' = \{e \in E : e \subseteq V''\}$.
 6:    If any edge $e \in E''$ has size $|e| \geq d + 1$, remove one arbitrarily chosen vertex in $e$ from $V''$.
       Let $V'$ denote the set of vertices after this removal step.
 7:    Run $I' = \text{BL}\,(H[V'])$; here, $H[V']$ denotes the subhypergraph of $H$ induced on $V'$.
 8:    Update $I = I \cup I'$, $V = V \setminus V'$.
 9:    for all $e \in E$ do
10:       if $e \cap (V' \setminus I') \neq \emptyset$ then
11:          $E \leftarrow E \setminus e$.
12:    for all $e \in E$ do
13:       $e \leftarrow e \setminus I'$.
14: Return $I$.
```

— Bercea et al., ACM T. Parallel Comp., 2016.

- We use **words** and **sentences**. It's mostly **self-contained**.
- Imperfections: (a) Line 1 should be part of the input. (b) Both "=" and "←" are used for assignment. (c) Mathematics separated by a comma.
- Should **reduce notation**: $I' = \texttt{BL}(\text{subhypergraph of } H \text{ induced on } V')$.

Pseudocode is not code



```
ALGORITHM 1: remap scheme for the $A_M^N$ H-Scale
Input:  $M$: number of data disks in original RAID;
        $N$: number of expanded disks;
        $addr_{sector}$: the address of a request;
        $L[\,]$: address boundary array;
Output: $R_n$: row ID of a $addr_{sector}$ after scaling;
        $D_n$: the requested disk's ID of a $addr_{sector}$ after scaling;
// get request's original physical location
$addr_{sector} \rightarrow (i, j)$;
// process (*): if request has not been migrated or is not migrated
if ($addr_{sector}$ in $[0, L[0])$) then
    // if request has not been migrated
    if ($i < M + N$ && $j \geq 1$) then
        $R_n = i$ ;
        $D_n = j$ ;
    end
    // if request has been migrated or is migrated
    else
        $R_n = i + [i/(M + N - 1)]N$ ;
        $D_n = j$ ;
    end
end
// process (**): if request in the newly generated RAID space
if ($addr_{sector}$ in $[L[0], L[1])$) then
    $R_n = (i/N)(M + N) + i \bmod (M + N) + j$ ;
    $D_n = j$ ;
end
```

— Wan et al., ACM T. Storage, 2016.

If we mean "and", we write "and" (not &&).

We don't need a semicolon ; to end a line. We write it to mean a semicolon.

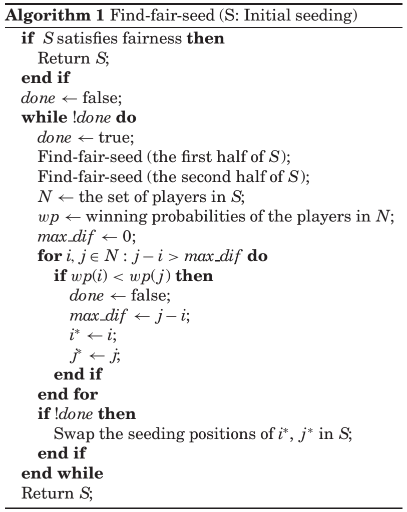
We don't write // to indicate a comment

Also

$$addr = a \times d \times d \times r.$$

Compare "*log*" ($log$) with "log" ($\log$).

Pseudocode is not code (cont.)



```
Algorithm 1 Find-fair-seed (S: Initial seeding)
if $S$ satisfies fairness then
    Return $S$;
end if
$done \leftarrow$ false;
while !$done$ do
    $done \leftarrow$ true;
    Find-fair-seed (the first half of $S$);
    Find-fair-seed (the second half of $S$);
    $N \leftarrow$ the set of players in $S$;
    $wp \leftarrow$ winning probabilities of the players in $N$;
    $max\_dif \leftarrow 0$;
    for $i, j \in N : j - i > max\_dif$ do
        if $wp(i) < wp(j)$ then
            $done \leftarrow$ false;
            $max\_dif \leftarrow j - i$;
            $i^* \leftarrow i$;
            $j^* \leftarrow j$;
        end if
    end for
    if !$done$ then
        Swap the seeding positions of $i^*$, $j^*$ in $S$;
    end if
end while
Return $S$;
```

— Vu and Shoham,
ACM T. Intell. Syst. Tech., 2011.

If we mean "not", we write "not" (not !).

Also ";" and $wp = w \times p$ etc., and minor typesetting problems.

I feel it's least ambiguous to use ← (typeset `\gets`) for assignment.
(Both "=" and ":=" alone are ambiguous.)
"Set $x = 2$" is also clear.

These look better:

```
\Call{Find-fair-seed}{the first half of $S$}
\Return
```

Phrases to learn and unlearn

If "different" means "various", then use "various"

> ... the cell sizes of **different** memory technologies are compared.
> — Sun et al., DATE, 2012.

"Different" reads awkwardly when used to mean "various". Ordinarily, we use "different" like this:

My hat is different to your hat.

Thus, in the top example, the reader stops and think:

...different memory technologies to what?

Afterwards, the reader realizes that "different" instead means "various", but it takes time; it's an obstacle to reading smoothly.

> ... the cell sizes of **various** memory technologies are compared.

Unlearn: “As shown in Table 1”, “As shown in Figure 1”

Continuing the previous example...

> **As shown in Table I**, the cell sizes of [various] memory technologies are compared.
>
> — Sun et al., DATE, 2012.

While this is grammatically correct...

- Tense mismatch: “Shown” (meaning “displayed”) is past tense, but “are” is present tense.
- Incorrect meaning: It does not say “[foo] are compared in Table 1”, but says “Table 1 shows that [foo] are compared”.

> The cell sizes of various memory technologies **are compared in Table I**.

> **We compare** the cell sizes of various memory technologies **in Table I**.

> **In Table I, we compare** the cell sizes of various memory technologies.

Unlearn: “As shown in Table 1”, “As shown in Figure 1”

There’s another problem after deducing that “shown” means “displayed” (not “demonstrated”):

> **As ~~shown~~ displayed in Figure 1**, the proposed algorithm uses a three-phase design.

The sentence is equivalent to this:

> The proposed algorithm uses a three-phase design **as displayed in Figure 1**.

Compare this to:

I use makeup as described in Girlfriend magazine.

Here, “Girlfriend magazine” determines how I wear makeup (not the other way around).
Likewise, the above implies Figure 1 above determines how the algorithm is designed. The logic is backwards!

Learn: “X increases as Y increases”, etc.

> We **can** observe that the MTTDL **drops dramatically** with the increase of the **amount** of hard drives.
>
> — a paper I reviewed (Chinese authors)

- *Problem 1*: Replace “we **can** observe” with “we observe”. Vagueness.
- *Problem 2*: You can break up with your girlfriend **dramatically**. Mobile phones have **dramatically** affected our lives. Inappropriate or over the top! (If you must exaggerate, then write “significantly”.)
- *Problem 3*: What do you mean it **drops**? Can we pick it up again? (Should be “decreases”.)
- *Problem 4*: The **amount** of hard drives? 50 kilograms? (Should be “number”.)
- *Problem 5*: I’ve seen a zillion awful ways of rephrasing “X increases as Y increases”; this is one.

> We observe that the MTTDL decreases as the number of hard drives increases.

Be careful with the word “doing”

Sometimes, “doing” is what a child says, whereas an adult is expected to be more precise.

> Trinity is an all-in-memory system, the graph data is loaded in memory before **doing** computation.
>
> — Shao et al., SIGMOD, 2013.

> Trinity is an all-in-memory system, the graph data is loaded in memory before **performing** computation.

> By **doing** this, we do not have to perform a post-processing step to process computationally-expensive verification.
>
> — Lu et al., ACM T. Info. Sys., 2014.

> **In this way, we avoid performing** a post-processing step to process computationally-expensive verification.