org-incremental

Daniel

August 16, 2021

Contents

`http://www.literateprogramming.com/`

This package is designed to bring incremental writing, incremental mail processing and incremental literate programming capabilities to org-mode and Emacs.

It should work with org-ql in order to find and generate lists of included files.

Files can be included with a `#+incremental_writing: t` header.

Key functions we would need:

- A-factor
- Priority
- Postpone/Reschedule (Intra- and inter-day)
- Dismissing (of files and subheaders)
- Mercy
- TODO keywords: ("to write", "to expand", "to rewrite", "to review", "finished")

- Deadline awareness (modify priority/a-factor/scheduled date based on distance to deadline)
- Statistical analysis to chart the progress through the writing material (`magit` and `calendar`)
- Outstanding headline sorting by priority to match circadian cycles (high for high alterness and low for end of day)

1 TODO Why incremental writing?

There are various purported benefits of incremental writing.

2 Topic spacing algorithm in elisp

Some documentation for the incremental writing algorithm can be found at:

- `https://help.supermemo.org/wiki/Creativity_and_problem_solving_in_SuperMemo#Incremental_writing_algorithm`
- `https://supermemopedia.com/wiki/SM_Algorithm_for_topics_%3F`
- `http://supermemopedia.com/wiki/How_was_the_topic_algorithm_created%3F`
- `http://supermemopedia.com/wiki/ABC_of_incremental_reading_for_any_user_of_spaced_repetition`
- `https://supermemo.guru/wiki/A-Factor`

Existing SRS algorithms in Emacs:

- `https://github.com/emacsmirror/org-contrib/blob/master/lisp/org-learn.el`
- `https://gitlab.com/phillord/org-drill`
- `https://github.com/l3kn/org-fc`

The basic algo is here:

```
(Interval=OldInterval*AFActor)
```

- The first metric is self explanatory, but `A-factor` [1] (standing for *absolute difficulty factor*) is more complicated in that it is used in older versions (<SM18) of Supermemo to represent item difficulty. It is still used for topics but not items in the current version.

The base value for `A-factor` in Supermemo is `2`, and so in essence the algo is simply a doubling mechanism:

```
(defcustom a-factor 2.0
  "Base a-factor value as per Supermemo defaults"
  :type 'float
  :group 'org-incremental)
```

As review spacings this is a simple geometric sequence ($x_n = ar^{(n-1)}$) with `2` as the common ratio:

These results are then sorted by priority, a user defined variable at the core of both incremental reading and writing. It should be noted that a key tool in the process is occasionally micromanaging interval lengths, which might grow at an undesirable rate for important articles and thus needs to be manually shortened from time to time.

In the functional style the interval determining algorithm:

- We use `round` here because human work days are measured in real days, which means we have a full circadian cycle between reps.

```
(defun determine-next-interval (old-interval a-factor)
  "Calcuate new interval for current headline.
Uses: (Interval=OldInterval*AFActor)"
  (let ((next-interval (* old-interval a-factor)))
    (round next-interval)))
```

Apply the base algorithm to existing `:PROPERTIES:` keys and then write the new interval:

[1]:: As it stands the value of the A-factor is not necessarily optimised to make use of the spacing effect. By Woz's own admission the current topic algorithm mostly serves as an obsolescence protocol, to push articles further and further out, and thus relies on user intervention in the form of modifying priorities (this is in-line with the current model) and micromanaging interval rescheduling. The latter is not too painful but we could likely be smarter about this.

```
(defun org-incremental-smart-reschedule ()
  (interactive)
  (let* ((old-interval (org-entry-get (point) "OLD_INTERVAL"))
         (a-factor (org-entry-get (point) "A-FACTOR")))
    (setq new-interval (apply 'determine-next-interval ;; not sure if apply is the be
                              (mapcar #'string-to-number `(,old-
interval ,a-factor))))
    (org-entry-put (point) "NEW_INTERVAL" (prin1-to-string new-
interval))))
```

```
*** [#15] example item
,SCHEDULED: <2020-02-17 Mon>
,:PROPERTIES:
,:ID:       dd92c87d-4407-4938-8472-a06b3882f7aa
,:A-FACTOR: 2
,:OLD_INTERVAL: 3
,:TOTAL_REPEATS: 8
,:LAST_REVIEWED: [2020-02-13 Thu 12:44]
,:NEW_INTERVAL: 4
,:END:
```

Is the `NEW_INTERVAL` value then used adjust the header's SCHEDULED date?

Next we need to write the previous `NEW_INTERVAL` to `OLD_INTERVAL`

We need to introduce checks for valid A-factor and interval values.

```
(assert (>= 2 2))
```

We can piggy back off of some more `org` functions:

- `org-default-priority` (30 in this case, with min being 60 and max 1)
-

3 org-ql prototyping

Let's create a test function to start bringing up a agenda-like view of tasks:

4 Test bed

4.1 [#30] example item

This is a test IW item.

```
*** [#B] example item
SCHEDULED: <2020-02-17 Mon>
:PROPERTIES:
:ID:       dd92c87d-4407-4938-8472-a06b3882f7aa
:A-FACTOR: 1.3
:OLD_INTERVAL: 3
:TOTAL_REPEATS: 8
:LAST_REVIEWED: [2020-02-13 Thu 12:44]
:NEW_INTERVAL: 4
:END:
```

4.2 UI