Group Rotation Type Crowdsourcing

Katsumi Kumai  
University of Tsukuba  
katsumi.kumai.2015b@mlab.info

Yuhki Shiraishi  
Tsukuba University of Technology  
yuhkis@a.tsukuba-tech.ac.jp

Jianwei Zhang  
Tsukuba University of Technology  
zhangjw@a.tsukuba-tech.ac.jp

Hiroyuki Kitagawa  
University of Tsukuba  
kitagawa@cs.tsukuba.ac.jp

Atsuyuki Morishima  
University of Tsukuba  
mori@slis.tsukuba.ac.jp

Abstract

A common workflow to perform a continuous human task stream is to divide workers into groups, have one group perform the newly-arrived task, and rotate the groups. We call this type of workflow the group rotation. This paper addresses the problem of how to manage Group Rotation Type Crowdsourcing, the group rotation in a crowdsourcing setting. In the group-rotation type crowdsourcing, we must change the group structure dynamically because workers come in and leave frequently. This paper proposes an approach to explore a design space of methods for group restructuring in the group rotation type crowdsourcing.

1 Introduction

Continuous human task streams appear in many applications, such as the captioning of real-time broadcasting and the metadata labeling to objects in videos [Lasecki et al. 2012] [Naim et al. 2013]. An example of task in such a task stream is to transcribe one spoken sentence into text.

Since human resources are limited, a common workflow to perform such a task stream is to divide workers into groups, have one group perform the newly-arrived task, and rotate the groups [WFD and WASLI]. In general, more than one worker belongs to each group for improving the result quality. We call this type of workflow the group rotation.

Fig. 1 illustrates a group rotation. Assume that we have a task stream for transcribing sentences spoken in a video. We have three groups $g_1$, $g_2$ and $g_3$. At present, workers in $g_1$ are performing the task. Each task asks workers to transcribe one sentence. Their results in a group will be aggregated for improving the task result by some means (e.g., majority voting). Then, workers in $g_2$ will transcribe the next sentence.

There are two points here. First, it is important to let workers know when their turn comes. Therefore, we put a counter on the task screen of each worker that countdowns until the sentence the worker has to transcribe appears. With the counter, workers can prepare for their turn.

Second, there is an application-specific number $d \geq 1$, which is the minimum number of workers in each group. In Fig. 1 $d = 2$. Usually, the minimum number of workers in a group is determined by the way how the application aggregates the answers to maintain the quality of task results.

**Group Rotation Type Crowdsourcing.** This paper addresses the problem of how to manage Group Rotation Type Crowdsourcing (GRTC), the group rotation in a crowdsourcing setting. Our assumption is that we can always recruit workers during the task stream, allowing workers to come in and leave freely. An example is to recruit volunteer workers from the audience of a lecture for transcribing the lecture.

Under the assumption, we must change the group structure dynamically because workers come in and leave while tasks are being performed. While it is desirable to increase the number of groups to make the burden to workers small, we must reduce the number of groups when there is a group having less than $d$ workers.

However, changing the group structure will give workers psychological stress, such as surprise, confusion or irritation. For example, if the counter jumps from 20 to 2, the worker would be surprised and feel stressed since she may not have prepared for the task. There is a clear trade-off between optimizing the number of groups and keeping the psychological stress of workers small. The problem may look similar to those for tree-form database indices such as B-Trees [Comer 1979] [Bayer and McCreight 1970]. In such index structures, we usually address the trade-off between the access time and the required space for storing the index. In contrast, a unique point of our problem is that the target is humans and not data. We address the trade-off between optimizing the number of groups and keeping the psychological stress given to workers small.

This paper proposes an approach to explore a design space of methods for group restructuring in GRTC. Our purposes are to (1) confirm the tradeoff between increasing the number of groups and keeping the psychological stress of workers caused by the move to other groups small, and (2) hopefully find sweet spots in the tradeoff.
2 Group Rotations

A group rotation state (or shortly a grs) is a building block of a group rotation and represents a snapshot of it. Fig. 1 illustrates a grs. Formally, a grs is defined as follows:

**Definition 1** A group rotation state S is defined as a tuple (W, G, Wg, Succ, p) where:

- W = {w₁, w₂, ..., w_n} is a set of workers. |W| = 9 in Fig. 3.
- G = {g₁, g₂, ..., g_m} is a set of groups |G| ≥ 2. G = {g₁, g₂, g₃} in Fig. 4.
- W_g : G → 2^W maps each group to the workers who belong to it. For example, |W_g(g₁)| = 3 in Fig. 5.
- Succ : G → G defines the next group of each group for the rotation. Succ(g₁) = g₂ in Fig. 7. The function is illustrated by direct edges among nodes representing groups.
- p ∈ G is the group whose workers are performing a task at this state. p = g₁ in Fig. 2.

In S, (1) every worker must belong to exactly one group, (2) each group has to have at least one worker, and (3) the graph must have a circle shape. Namely, S must satisfy all the following conditions:

1. W_g(g₁) ⊕ W_g(g₂) ⊕ ... ⊕ W_g(gₘ) = W,
2. ∀g_x ∈ G (|W_g(g_x)| ≥ 1), and
3. For any two groups gᵢ, gⱼ ∈ G, there is one and only the path from gᵢ to gⱼ with Succ.

Given two grs’s S and S’ we say that S’ follows S if any worker in p who performed tasks in S does not perform any task in S’ and p’ is the successor of p in S’. Formally,

**Definition 2** Let S and S’ be group rotation states, and let p and p’ be the current groups of S and S’, respectively. We say S’ follows S if (1) W_g(p) ∩ W_g(p’) = φ, (2) p exists in G’, and (3) p’ = Succ’(p).

A group rotation is a sequence of group rotation states each of which follows its predecessor.

**Definition 3** Let [S₁, S₂, ...] be a sequence of group rotation states. The sequence is a group rotation if for any successive pair (Sᵢ, Sᵢ₊₁) in the sequence, Sᵢ₊₁ follows Sᵢ.

3 Group Rotation Generators

Assume that we have an application-dependent minimum number of workers for each group (denoted by d), a sequence T = [t₁, t₂, ...] of times when each task is performed, and a sequence ΔW = [Δw₁, Δw₂, ...,] of worker changes. Here, Δwᵢ is either +wᵢ (i.e., wᵢ comes in) or −wᵢ (wᵢ leaves) and has a property t(Δwᵢ) to represent the time when the change happens. Then, we can generate a group rotation with a group rotation generator, an algorithm to generate group rotations.

A group rotation generator is defined as follows:

**Definition 4** The group rotation generator is defined as a function Next : States × Int × Diff f → States that takes as input Sᵢ, d and a subsequence of ΔW and generates the next Sᵢ₊₁ s.t. Sᵢ₊₁ follows Sᵢ.

Algorithm 1 Template for Next(Sᵢ, d, ΔWᵢ)

**Input:** Sᵢ, d, ΔWᵢ

**Output:** Sᵢ₊₁

1: S_tmp ← Sᵢ
2: for Δwⱼ ∈ ΔWᵢ do
3: if Δwⱼ is +wⱼ then
4: S_tmp ← Insert(S_tmp, d, +wⱼ)
5: else if Δwⱼ is −wⱼ then
6: S_tmp ← Remove(S_tmp, d, −wⱼ)
7: end if
8: end for
9: Sᵢ₊₁ ← S_tmp with pᵢ₊₁ = Succᵢ₊₁(pᵢ).

Given (Sᵢ, d, T, ΔW, Next), the following procedure generates a group rotation.

1. Output S₁ as the first grs in the group rotation.
2. At each tᵢ in T do the following.
   (a) Let ΔWᵢ be the subsequence of ΔW in which t(Δwⱼ) for each Δwⱼ ∈ ΔWᵢ is in (tᵢ₋₁, tᵢ]. Namely, ΔWᵢ is a set of worker changes from tᵢ₋₁ to tᵢ.
   (b) Sᵢ₊₁ = Next(Sᵢ, d, ΔWᵢ) where pᵢ₊₁ = Succᵢ₊₁(pᵢ).

Algorithm 1 shows a design space for Next(Sᵢ, d, ΔWᵢ). In the design space, we apply two worker-at-a-time update operators (named Insert and Remove) for generating Sᵢ₊₁ in a sequential way according to worker insertion and deletion described in the sequence ΔWᵢ. The algorithm works as follows. First, it copies Sᵢ to S_tmp (Line 1). Next, it applies Insert and Remove operators with each worker Δwⱼ in ΔWᵢ (Lines 2 to 8). Finally, it copies S_tmp to Sᵢ₊₁ and moves the current group forward (Line 9). The two operators work as follows. First, Insert(S, d, +wⱼ) chooses a group and inserts wⱼ into it. If the number of workers in the group is larger than a function of d (denoted by max(d)), it splits the group into two groups. Second, Remove(S, d, −wⱼ) first deletes wⱼ from a group. If the number of workers in the group becomes less than d, it moves other workers to the group if we find a group having many workers, otherwise joins the groups with another group to meet the condition.

Here, we see four key components: choose, split, find and join. We consider a variety of possible methods to implement the four components in Insert and Remove operators. For example, choosing a group into which we insert worker heavily affects how often the groups are restructured and how much worker stresses work.

**Acknowledgments**

The authors are grateful to the contributors to Crowd4U, whose names are publicly listed at http://crowd4u.org. This work was partially supported by JSPS KAKENHI (#25240012, #26870090, #16K16460), Collaborative Research Program at NII, Expense for Strengthening Functions in NTUT’s Budgetary Request for Fiscal 2016 and Promotional Projects for Advanced Education and Research in NTUT.
References

[Bayer and McCreight 1970] Bayer, R., and McCreight, E. M. 1970. Organization and maintenance of large ordered indexes. In Record of the 1970 ACM SIGFIDET Workshop on Data Description and Access, November 15-16, 1970, Rice University, Houston, Texas, USA (Second Edition with an Appendix), 107–141. ACM.

[Comer 1979] Comer, D. 1979. The ubiquitous b-tree. ACM Comput. Surv. 11(2):121–137.

[Lasecki et al. 2012] Lasecki, W. S.; Miller, C. D.; Sadilek, A.; Abumoussa, A.; Borrello, D.; Kushalnagar, R. S.; and Bigham, J. P. 2012. Real-time captioning by groups of non-experts. In Miller, R.; Benko, H.; and Latulipe, C., eds., The 25th Annual ACM Symposium on User Interface Software and Technology, UIST ’12, Cambridge, MA, USA, October 7-10, 2012, 23–34. ACM.

[Naim et al. 2013] Naim, I.; Gildea, D.; Lasecki, W. S.; and Bigham, J. P. 2013. Text alignment for real-time crowd captioning. In Vanderwende, L.; III, H. D.; and Kirchhoff, K., eds., Human Language Technologies: Conference of the North American Chapter of the Association of Computational Linguistics, Proceedings, June 9-14, 2013, Westin Peachtree Plaza Hotel, Atlanta, Georgia, USA, 201–210. The Association for Computational Linguistics.

[WFD and WASLI] WFD and WASLI. WFD and WASLI guidelines on securing and utilising the services of sign language interpreters for the United Nations (June 2015). https://www.wfdeaf.org/wp-content/uploads/2015/07/Interpreter-Guidelines-for-UN-24-Updated-June-2015.pdf. Accessed: 2016-08-06.