University of Zurich[UZH]

Patrick Minder
Abraham Bernstein

CrowdLang: programming human computation systems

TECHNICAL REPORT – No. IFI-2012.03

2012

University of Zurich
Department of Informatics (IFI)
Binzmühlestrasse 14, CH-8050 Zürich, Switzerland

ifi

Patrick Minder, Abraham Bernstein
CrowdLang: programming human computation systems
Technical Report No. IFI-2012.03
Dynamic and Distributed Information Systems
Department of Informatics (IFI)
University of Zurich
Binzmuehlestrasse 14, CH-8050 Zurich, Switzerland
http://www.ifi.uzh.ch/ddis/

CrowdLang - Programming Human Computation Systems

Interweaving Human and Machine Intelligence in a Complex Translation Task

Patrick Minder
University of Zurich
Dynamic and Distributed
Information Systems Group
minder@ifi.uzh.ch

Abraham Bernstein
University of Zurich
Dynamic and Distributed
Information Systems Group
bernstein@ifi.uzh.ch

ABSTRACT

Today, human computation systems are mostly used for batch processing large amount of data in a variety of tasks (e.g., image labeling or optical character recognition) and, often, the applications are the result of extensive lengthy trial-and-error refinements.
A plethora of tasks, however, cannot be captured in this paradigm and as we move to more sophisticated problem solving, we will need to rethink the way in which we coordinate networked humans and computers involved in a task. What we lack is an approach to engineer solutions based on past successful patterns.

In this paper we present the programming language and framework CrowdLang for engineering complex computation systems incorporating large numbers of networked humans and machines agents incorporating a library of known successful interaction patterns. CrowdLang allows to design complex problem solving tasks that combine large numbers of human and machine actors whilst incorporating known successful patterns.
We evaluate CrowdLang by programming a text translation task using a variety of different known human-computation patterns. The evaluation shows that CrowdLang is able to simply explore a large design space of possible problem solving programs with the simple variation of the used abstractions.

In an experiment involving 1918 different human actors we, furthermore, show that a mixed human-machine translation significantly outperforms a pure machine translation in terms of adequacy and fluency whilst translating more than 30 pages per hour and that the mixed translation approximates the human-translated gold-standard to 75% using the automatic evaluation metric METEOR. Last but not least, our evaluation illustrates that a new human-computation pattern, which we call staged-contest with pruning, outperforms all other refinements in the translation task.

Author Keywords

CrowdLang, Programming Language, Human Computation, Collective Intelligence, Crowdsourcing, CrowdLang, Translation Software, Pattern Recombination

A short research note summarizing this technical paper with the title "How to Translate a Book Within an Hour - Towards General Purpose Programmable Human Computers with CrowdLang" was published at *ACM WebSci Conference 2012*.

ACM Classification Keywords

H.1.2 User/Machine Systems : H.4.1 Workflow management

General Terms

Algorithms, Human Factors, Languages

1. INTRODUCTION

Much of the prosperity gained by the industrialization of the economy in the 18th century arose from the increased productivity by dividing work into smaller tasks performed by more specialized workers. Wikipedia, Google and other stunning success stories show that with the rapid growth of the World Wide Web and the advancements of communication technology, this concept of Division of Labour can also be applied on knowledge work [24, 23]. These new modes of collaboration— whether they are called collective intelligence, human computation, crowdsourcing or social computing[1] —are now able to routinely solve problems that would have been unthinkably difficult only a few years ago by interweaving the creativity and cognitive capabilities of networked humans and the efficiency and scalability of networked humans in processing large amount of data [5]. The advent of crowdsourcing markets such as Amazons Mechanical Turk (MTurk)[2], Clickworker[3], or CrowdFlower[4] even fosters this development and Bernstein et al. suggest that, as the scale and scope of these human-computer networks increase, we can view them as constituting a kind of a global brain [5].

Even though there exist hundreds of human computation systems that harness the potential of this "global brain", our understanding of how to *"program"* these systems is still poor because human computers are different from traditional computers due to the huge motivational, error and cognitive diversity within and between humans [5]. As a consequence, today, human computation is only used for massive parallel information processing for tasks such as image labeling or tagging. These tasks share in common that they are massively

[1]There is an ongoing debate in the research field about the clear distinction between these concepts [28, 17, 23]. In the context of this paper and in analogy to Law et al. [17], we simply consider *human computation* as computation that is carried out by humans and likewise the term *human computation systems* (*HCS*) describes "paradigms for utilizing human processing power to solve problems that computers cannot yet solve" [29].
[2]http://www.mturk.com
[3]http://www.clickworker.com/
[4]http://www.crowdflower.com/

parallelizable, have a low interdependence between single assignments in the task decomposition, and use relatively little cognitive effort.

A plethora of tasks, however, cannot be captured in this paradigm. Consider, e.g., the joint editing of lengthy texts as accomplished on Wikipedia. Here, a large number of actors work on highly interdependent tasks that would be very difficult to cast into a bulk parallelization with low interdependence. Hence, to harness the full potential of human computation systems, we need new powerful programming metaphors that support the design and implementation of human computation systems, as well as general-purpose infrastructure to execute them. Specifically, we need a programming language that supports the whole range of possible dependencies between single tasks, allows for the seamless reuse of known human computation patterns incorporating both human and machine operators to exploit prior experience, and integrates multiple possible execution platforms such micro task markets as well as games-with-a-purpose platforms to leverage a large ecosystem of participants. Furthermore, to move from a culture of *"wizard of oz"*-techniques, in which applications are the result of extensive trial-and-error refinements, this programming language has to support the recombination [4] of interaction patterns to systematically explore the design space of possible solutions.

Recent research only partially addresses these challenges by providing programming frameworks and models [21, 14, 1] for massive parallel human computation, concepts for planning and controlling dependencies [3, 33], and theoretical deductive analysis of emergent collective intelligence [23].

In this article, we present the human computation programming language and framework *CrowdLang* for interweaving networked humans and computers. *CrowdLang* supports cross-platform workforce integration, the management of latency caused by human computers, as well as incorporates abstractions for group decision, contest, and collaborative interaction patterns to exploit prior experience, as proposed by Malone et al. [23]. Furthermore, in contrast to several MapReduce inspired approaches [14, 1], *CrowdLang* supports the management of arbitrary dependencies among tasks and workers and not only synchronous parallelization. We illustrate *CrowdLang*'s feasibility and strength in interweaving human and machine actors by programming a collection of text translation programs. We show that the resulting translation programs are capable to speedily translate non-trivial texts from German to English achieving a significantly better quality than pure machine translation approaches. In addition, given the simple recombination of patterns supported by *CrowdLang*, we were able to unearth a novel human computation pattern called *"Staged-Contest with Pruning"* that outperforms all other known patterns in the translation task.

As such, the contributions of the paper are as follows: *(1)* We present *CrowdLang*. *(2)* We introduce the *CrowdLang* execution environment and its plug-and-use architecture for generic language extensions. *(3)* We provide the 'pattern recombinator' methodology – a translation of the process recombination approach [4] to human computation tasks. *(4)* We present a set of human computation programs that allow translating more than 30 pages per hour with a "good" quality when compared to professional translations. *(5)* Finally, we present a new human computation interaction pattern called "Staged-Contest with Pruning" that outperforms all other patterns in the translation task.

2. BACKGROUND AND RELATED WORK

In this section, we review the state-of-the-art in the research field of human computation. Especially relevant to this paper is research about frameworks for supporting automated execution of human computation *"programs"*, the design of novel interaction patterns, and several analysis of human computation systems.

2.1 Management and Programming Frameworks

Recently, the research community has proposed a number of programming frameworks and concepts that address the distinct challenges in engineering human computation systems. Little et al. [19, 21, 20] proposed the use of the imperative programming framework *TurKit*. In particular, investigating workflows composed by iterative and parallel traditional programming constructs, they explored basic technical problems caused by the high latency associated with waiting for a response from a human worker, writing, and debugging human computation code. Hence, they support the idea of a "crash-and-rerun" programming model, which allows a programmer to repeatedly rerun and debug processes without republishing costly previously completed human computation tasks. A similar approach is used in the crowdsourced database query processing system *CrowdDB* proposed by Franklin et al. [11].

Several programming frameworks inspired by the MapReduce [10] programming metaphor have been proposed to coordinate arbitrary dependencies between interdependent tasks. These frameworks model complex problems as a sequence of partition, map, and reduce subtasks. For example, Kittur et al.'s *CrowdForge* programming framework [14] starts by breaking down large problems into discrete subtasks either by using human or machine computers. Then human or machine agents are used to collect a set of solutions. Finally, the results of multiple workers are merged and aggregated into the solution of the larger problem. Built on top of the crowdsourcing market *CrowdFlower*[5], they also presented a visual approach for developing and managing these workflows by visually drag-and-dropping operators into a workflow model. Similarily, Ahmad et al.'s *Jabberwocky* programming environment [1] extended this idea by providing an additional human and resource management system for integrating workforces from different markets, as well as a high-level procedural programming language. Finally, Noronha et al. [26] suggest a divide-and-conquer management framework inspired by corporate hierarchies.

These studies highlight the importance of designing new environments for programming human computation systems but still are limited in the restrictions to the structural, synchronous rigidness of the MapReduce programming

[5]http://crowdflower.com/

metaphor when modeling workflows with arbitrary dependencies [22]. Further, they lack in providing any explicit treatment of cognitive diversity in and between human actors [5], the lack of abstractions for complex coordination patterns such a group decision procedures [23], and assume that computation can be fully specified ex-ante. In particular, many complex problem-solving processes are difficult to specify ex-ante and only gain more specific definitions during execution, when the problem to be solved clarifies, or may start out as well defined tasks and then lose their specific definition due to some unexpected exceptions. Thus, we proposed the notion that processes move along a *specificity frontier* from well-defined and static to loosely defined and dynamic [3]. To harness the full potential of human computation systems, we believe that programming languages designed for this purpose should exhibit all these features.

Furthermore, several systems are proposed that exploits self-organizing crowds to solve a planing under constraints problem [33], to decompose large search queries [18, 32], or to iteratively decompose a task in finer-grained subtasks [15, 16]. These systems illustrate the crowd-based solutions of a completely different coordination problem than the ones introduced above. It is, however, not a general-purpose approach suitable to most problems.

2.2 Interaction Patterns

In analogy to software engineering design patterns human computation researchers look for generalizable human computation patterns in various fields. Bernstein et al. [**?**], for example, suggests the use of the Find-Fix-Verify pattern for qualitative crowd sourced proofreading. Zhang et al. [9] an iterative dual pathway pattern for speech-to-text transcription. These studies suggest that programming human computation system is likely to incorporate a number of such patterns.

2.3 Deductive Analysis of Human Computation Systems

Complementing these almost empirical explorations of possible patterns several studies [28, 17, 30, 7, 12] taxonomize various aspects of human computation systems.

In addition, based on an examination of 250 different human computation systems, Malone et al. [23] provide a deductive analysis of collective intelligence and human computation systems. In particular, their *Collective Intelligence Genome* identified the characteristics (*"genes"*) that are recombined to the basic building blocks (*"genome"*) in those applications. Their conceptual classification framework suggests to characterize each building block by using two pairs of related questions. First, they considered staffing *(Who is performing the task?)* and different kind of incentives *(Why are they doing it?)*. Second, they analyzed a specific system by defining the goal of a task *(What is being done?)* and problem-solving process *(How is it being done?)*.

Malone et al.'s classification framework is particularly relevant to this paper because it supports the idea that human computation systems can be built by recombining those basic building blocks as Bernstein [4] also proposed in the context of business processes.

3. CROWDLANG

In this section we introduce *CrowdLang* [25]— our human computation programming language and framework. We will first present the design goals. This will be followed by a discussion of the system's architecture, the programming language, and its implementation of the collective intelligence genes.

3.1 Key Ideas and Design Goals

Conventional programming languages and framework are developed to interoperate with deterministic machines and, thus, provide only a limited set of sophisticated abstractions for coordinating the behavior of a program (e.g., loops or conditionals). When moving from programming pure machine computation to hybrid machine-human or pure human computation systems these languages are not a good match due to the cognitive, error, and motivational diversity within and between humans [5] and the varying degrees of detail in many human task definitions. Additionally, the proposed human computation frameworks (see Section 2.1) are also restricted by the structural rigidness of the MapReduce programming metaphor.

The objective of *CrowdLang* is to build a general-purpose programming language and framework for interweaving human and machine computation within complex problem solving. *CrowdLang* intends to incorporate explicit methods for handling (cognitive, error, and motivational) diversity, complex coordination mechanisms (and not only batch processing) and abstractions for human computation tasks such as group decision processes. In future version, the framework will also support the specificity frontier to allow unstructured, constraint-restricted computation and for run-time task decomposition as well as the modeling of of non-functional constraints such as budget, completion time, or quality. Last but not least, the *CrowdLang* engine has to address the technical challenges associated with crowd worker latency (waiting for the response by humans) [19]. In designing the first prototype of *CrowdLang* we pursued the following design goals:

(1) Interweaving human and machine intelligence: CrowdLang simplifies the management of computation algorithms incorporating human and machine actors. It provides a model-based programming language, an application programming interface for the asynchronous integration of human computation, and provides abstractions for the management of a wide range of dependencies between algorithmic elements.

(2) Simplify the design of new human computation algorithms through the seamless reuse of existing interaction patterns and pattern recombinations: The *CrowdLang* framework limits the effort for the development of ad-hoc infrastructure (e.g., work list management, cross-platform workforce integration or latency management). Additionally, *CrowdLang* actively supports the exploration of the design space of human computation algorithms solutions through (1) the seamless integration of existing interaction patterns[6] and (2) pat-

[6]see http://www.ifi.uzh.ch/ddis/research/CrowdLang.html for a catalogue of typical human computation interaction patterns in CrowdLang

tern recombinations as proposed for business process management [4]. Also, *CrowdLang* is extensible through the use of a plug-in infrastructure.

(3) Support of cross-platform workforce integration and the management of crowd worker activities: Additionally, *CrowdLang* fosters cross-platform human computation and limit the drawbacks of crowd worker anonymity by integrating different type of workforce sources and tracking crowd worker histories. Using the latter, we enhance the allocation (or routing) of new task to skilled workers and limit the amount fraudulent behavior within crowd markets.

3.2 Architecture

The CrowdLang framework architecture (see Figure 1) consists of three main components: the *IDE*, the *Engine* and *Integrator*.

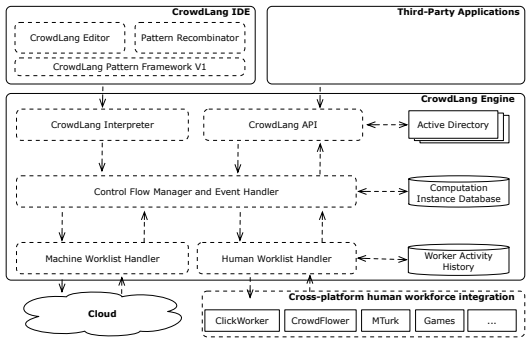

Figure 1. CrowdLang Architecture

The *CrowdLang IDE* provides a source code editor for the design of human computation algorithms. In a future version, we will also integrate the *CrowdLang pattern recombinator*, which simplifies the automated exploration of the design space through pattern recombination.

The *CrowdLang Engine* is responsible for managing the execution of *CrowdLang* algorithms. The engine consists of five different components. *(1)* The *Interpreter* translates the problem statement (source code) into an executable control flow. *(2)* The *API* allows to call single operators from a third-party application using Web service calls. *(3)* The *Control Flow Manager* manages the problem solving process by handling incoming events (e.g. results from the human worklist handler) and routing the problem solving process. Furthermore, it manages different types of events such as finished assignments by crowd worker or exceptions by storing the state of the problem solving process in the *Computation Instance Database*. Similar to the *Crash-and-Rerun* programming model [21] it ensures error recovery after exceptions. *(4)* The *Human Work-list Handler* manages the (asynchronous) interactions with the crowd: publishing new tasks to the crowd (e.g. to MTurk), providing resources for the task (web interface), and recognizing finished tasks. *(5)*, Finally the *Machine Work-list Handler* manages the interaction with computational operators such as Web services.

To conclude, at the bottom level, the *CrowdLang Integrator* builds the interface to different human workforce crowds and machine operators in the cloud.

3.3 CrowdLang - A Programming Language

In accordance with Malone et al.'s [23] empirical exploration of emergent collective intelligence, the *CrowdLang* programming language provides operators for the task decomposition, the management of producer-consumer relationships and resource allocation, as well as the seamless reuse of known human computation patterns.

3.3.1 Fundamental Operators of CrowdLang

CrowdLang provides language constructs for defining basic operators (performed by humans or machines), data items, and control flow constructs (see Figure 3). A *Task* represents the transformation of a given problem statement into a solution. The transformation is performed either by humans *(Human Computation)* or machines *(Machine Computation)*. A *Problem Statement* defines a task in terms of a question and the required input data. A *Solution* represents the computed results for a *Problem Statement*. Finally, a *Sequence Flow* defines the execution order of single tasks and manages therefore classical producer/consumer relationships in the form of a prerequisite constraint, where the produced output of a previous task is consumed as an input in the next task.

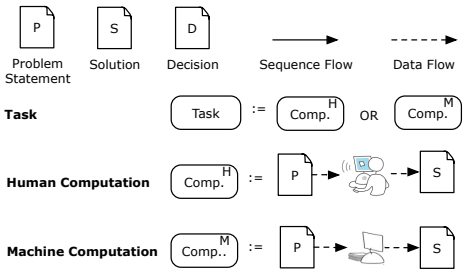

Figure 2. Fundamental CrowdLang Operators

3.3.2 Task Decomposition and Distribution

To enable more sophisticated problem-solving *CrowdLang* provides a set of routing operators to distribute computation and aggregating results (see Figure **??**). The *Divide-and-Conquer* operator decomposes a problem statement into multiple parallelizable, distinct subproblems. The *Aggregate* operator, in contrast, aggregates the results of several subtasks to a solution of the initial problem statement.

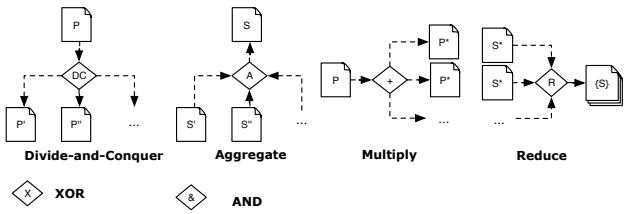

Figure 3. Routing, Aggregation, and Task Decomposition

A given problem can be distributed to actors in three different ways. The *Multiply* control flow operator indicates that a given problems gets transformed multiple times in parallel. Hence, copies of the original problem statement get allocated to multiple independent actors potentially leading to different solutions (in particular when perfumed by human actors). Hence, the result of such an execution is a set of solutions. The *Reduce* operator takes a set of solutions and determines the "best" solution candidate employing a decision procedure. Together the *Multiply* and *Reduce* are the building block for many parallelizing crowd computing patterns.
In addition, *CrowdLang* provides the established *exclusive (XOR)* and *parallel (AND)* control flow operators. XOR is used to create or synchronize alternative paths; AND can be used to create and synchronize parallel paths.

3.3.3 Building Blocks of Collective Intelligence

Based on the previously defined operators, *CrowdLang* defines a set of basic building blocks for interweaving human and machine computation in the *CrowdLang* framework. In accordance to [23], we distinguish *Create* and *Decide* interaction patterns.

Create Interaction Patterns *CrowdLang* defines two variations of create interaction patterns: Collection and Collaboration.

A *Collection* occurs when actors independently contribute to a task. Malone et al. [23] illustrated the Collection in terms of posting videos on YouTube.In *CrowdLang* a Collection is defined as a multiplied independent transformation of a problem statement into a proposed solution using the *Multiply* operator. *CrowdLang* defines two variations of the Collection gene. First, A *Job* (see Figure 4) is a simple *Multiply-AND* combination resulting in a set of solutions.

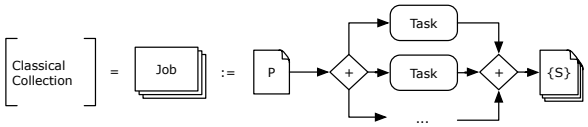

Figure 4. Classical Collection

Second, a *Contest* (see Figure 5) is a *Job* followed by a *Reduce* selecting the *Job*'s best solution based on a decision.

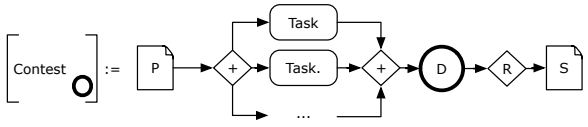

Figure 5. Contest

A *Collaboration* occurs when actors cooperate by contributing either iteratively or by solving different parts of a problem. *CrowdLang* supports two variations of the collaboration gene (see Figure **??**C/D). First, an *Iterative Collaboration* models problem solving as an *iterative process of interdependent solution improvement* whereas the submitted contributions are strongly interdependent on previous ones. It can be likened to the `repeat ... until <condition>` construct of a typical programming language. A typical example of this process is article writing in the Wikipedia, iterative labeling, or OCR [**?**]. Based on a problem statement a crowd worker builds an initial version of the solution followed by a decision process where either the crowd or a machine decide whether the proposed solution needs further refinement. This procedure will be repeated until the decision procedure accepts the solution.

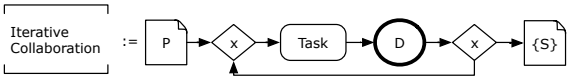

Figure 6. Iterative Collaboration

Second, a *Parallelized Interdependent Subproblem Solving* represents the combination between a divide-and-conquer of the initial problem, the parallel execution of the partial problems, and the aggregation of the results to the final solution. The main advantage of this pattern is that it allows to first split a problem into a set of independent subproblems that can then be solved in parallel. Open source programming is an example for this pattern, where an overall problem specification is divided into subsystem, each of which are programmed and then linked together to build the resulting system.

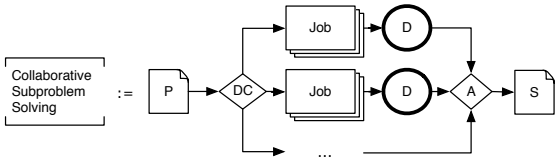

Figure 7. Parallelized Interdependent Subproblem Solving

Decide Interaction Patterns *CrowdLang* currently supports both group and individual decision. A *Group Decision* is defined as a mechanism that determines the best solution by using multiple crowd worker in an independent manner. Examples of group decisions are the evaluation of different solutions by voting, forced agreement, or parallel guessing with aggregation. An *Individual Decision* is a decision is the result of an evaluation by a single human or machine agent. Note that these specifications depart from Malone et al.'s framework. There a group decision is defined as a decision that a group make that subsequently holds for all participants (e.g., elections, ballot questions for new laws, etc.). Our operationalization allows for group based decisions that affect only individual actors or affect all individuals differently exemplified by the use of a recommendation system to aid movie selection.

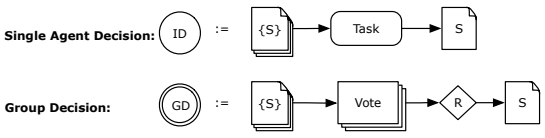

Figure 8. The Decide Gene: Single Agent and Group Decisions

4. DESIGN A NEW APPLICATION WITH CROWDLANG

In this section we show how to program a new application with *CrowdLang* by interweaving networked humans and computers. Therefore, we illustrate how *CrowdLang* supports high-level pattern recombination by developing a family of 9 non-trivial text translation programs.

4.1 Translating Texts with CowdLang

Using *CrowdLang*, we developed a prototype of a text translation engine incorporating human crowd worker and machine translation. In the development process we used the *CrowdLang Library* and *Intelligent Assistant* for recombining different workflow refinements. The development process included the following five steps:

1. *Identify the Core Activities:* A programmer starts with the definition of an abstract solving algorithm by identifying abstract core activities (operations) and Producer-Consumer dependencies [22] among them.

2. *Define the Design Space:* Then, (s)he selects a set of suitable interaction patterns from the *CrowdLang Library* which can be applied as operators for the abstract core activities.

3. *Generate the Recombinations:* Then the *Intelligent Assistant* systematically generates a set of alternative refinements by recombining the selected patterns from the *CrowdLang Library*.

4. *Execution:* The programmer executes the alternative refinements.

5. *Evaluation:* Finally, (s)he evaluates the generated variations and selects the best algorithm among the set of alternative refinements.

In the following, we illustrate some of the steps in the context of the translation task in more detail.

4.1.1 Identify core activities

We started by defining an abstract problem-solving workflow for the translation task and modeled the core activities and producer-consumer dependencies among them in Figure 9. This workflow starts by first iteratively splitting the input—an article—into paragraphs and then sentences (`Task Decomposition`); then processes the resulting sentences in parallel by sequentially applying machine translation (`MT`) and crowd-based rewriting (`Rewrite`) ; The using an aggregate operator (`A`) the sentences are combined to paragraphs that are then assigned to crowd-workers to improve the language quality by enhancing paragraph transitions and enforcing a consistent wording (`Improve Language Quality`). Finally, the grammatical correctness is improved (`Check Syntax`) by eliminating syntactical and grammatical errors.

4.1.2 Define the Design Space and Generate the Recombinations

We systematically generated a set of 9 alternative refinements for the algorithm by recombining existing patterns. For each refinement we chose of 3 patterns for both `Rewrite` and `Improve Language Quality`. The chosen patterns (all shown in Figure 9B) were (1) Contest with Six Sigma Pruning, (2) Iterative Improvement, and (3) an Iterative Dual Pathway Structure first presented in the context of Speech-to-Text transcription [9]. Further, we used an adaption of the spell checking pattern Find-Fix-Verify [6] for `Check Syntax`.

Contest with Six Sigma Pruning (CP) uses an adapted contest interaction pattern for generating semantical correct sentences and improve text quality (see Figure 5). First, 3 different worker generate solutions. Then, these proposed solutions are pruned using the six sigma rule [8, p. 320 - 330]. The six sigma rule—a statistical method originally used in operations research—intends to improve the output quality of a process by minimizing variability. Specifically, we compared the crowd-workers' working time on a task compared to a previously collected average. Defining the average work time as $\widehat{w}$ we hypothesizes that tasks should be accomplished within the interval $\widehat{w} \pm 3\sigma$ with σ as the standard deviation of the normal distribution. We minimize the number of "lazy turkers" (someone who tries to maximize his earnings by cheating) by rejecting results of workers when the working time is shorter than the lower bound $\widehat{w} - 3\sigma$. We also identify so called "eager beavers" (people, who are going beyond the task requirements) with an upper bound on execution time of $\widehat{w} + 3\sigma$. Finally, we select the best solution among the remaining ones using a group decision. In particular, we ask 5 workers to rank the proposed solutions and then apply the borda rule [31] to determine the winning solution.

Iterative Improvement (II) uses a iterative collaboration interaction pattern for generating semantical correct sentences and improve text quality (see Figure 6). We define three termination conditions: (1) two out of three crowd worker assess a sentence as semantically correct, (2) the result of an iteration step is equivalent to the previous one, or (3) we exceed the number of three iterations.

Iterative Dual Pathway Structure (DP) is an adaptation of [9]. Here we assign the same problem (e.g., an initial translation) to two different paths (see Figure 10). In each of the two paths a worker is asked to improve a given translation $Comp^{H1}$ and $Comp^{H2}$. At the end of this step the solution of the two paths are compared. If the two solutions are equivalent based on an individual decision by a third crowd-worker then we have a final result. Else, we iterate by sending each of the results back along its path for additional improvements until they are judged equivalent.

Find-Fix-Verify (FFV) checks the grammatical and syntactical correctness of a text fragment by first ask crowd worker to find misspellings and grammatical errors. Then a group of

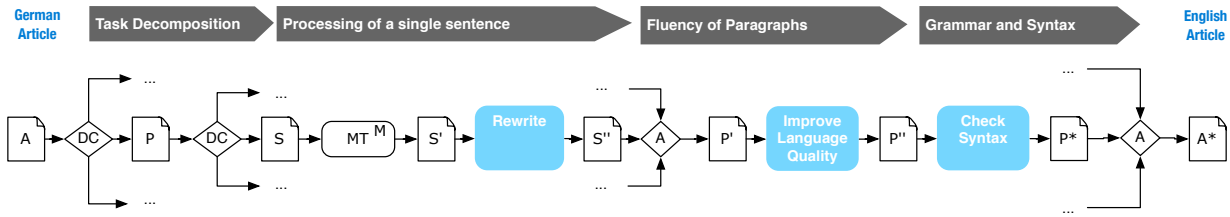

Figure 9. Abstract translation algorithm

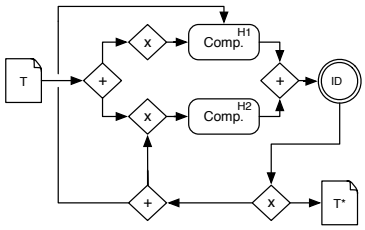

Figure 10. Iterative Dual Pathway Structure (DP)

crowd worker is asked to propose a solution for the identified problems. Finally, the solutions are verified by three independent crowd worker (see Figure 11). Additionally, we adapted this pattern slightly by introducing also a spell checking software $Comp^M$.

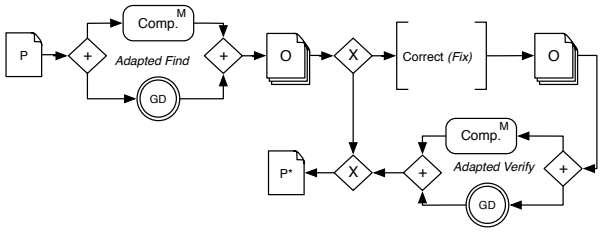

Figure 11. Find-Fix-Verify (FFV)

The resulting 9 alternative refinements are presented in Table 1.

	Rewrite	Improve L. Quality	Check Syntax
CP x CP	CP	CP	FFV
CP x II	CP	II	FFV
CP x DP	CP	DP	FFV
II x CP	II	CP	FFV
II x II	II	II	FFV
II x DP	II	DP	FFV
DP x CP	DP	CP	FFV
DP x II	DP	II	FFV
DP x DP	DP	DP	FFV

Table 1. Resulting pattern recombinations

4.2 Evaluation

We evaluated the different translation algorithms implemented with *CrowdLang* along a number of dimensions. We compare the results of the 9 runs as well as a pure machine translation with a gold standard human translation using automatic text analysis measure. The best two program combination additionally get compared to the gold standard by the crowd as well as professional translators.

4.2.1 Experimental Setup

The evaluation was conducted on a standard German to English translation task. Specifically, we generated translations for 15 different articles from Project Syndicate[7]— a Web source of original op-ed commentaries —totaling in 153 paragraphs with 558 sentences and 10'814 words translated from German to English. As a baseline we considered Google Translate[8].

Evaluation Aspects We analyzed the resulting translations in terms of performance and quality. First, we considered different performance metrics such as average work time, throughput time (including also the waiting time), and the costs per sentence. Second, based on literature research [27], we judged the translation quality by comparing each evaluation segment (e.g, a paragraph) with a high quality reference translation along three different dimensions:

1. *Adequacy:* The meaning by the reference translation is also conveyed by the output of a translation algorithm
2. *Fluency:* The translation being evaluated is judged according to how fluent it is without comparing it against a reference translation.
3. *Grammar:* A translation segment is being evaluated according to its grammatical correctness without comparing it against a reference translation.

Evaluation Methodology The crowd-based translation processes were evaluated using automatic machine, non-professional, and professional human evaluation.

Automatic Evaluation First, we approximated the translation quality with the METEOR [2] score, which automatically estimates human judgment of quality using unigram matching between a candidate and reference translation. In accordance with its standard usage we report the METEORs cores without using WordNet[9] synonyms to match candidate and reference translations. We considered one reference translation for each evaluation segment. Hence, a translation attains a score of 1 if it is identical to the reference translation.

[7]http://www.project-syndicate.org/
[8]http://translate.google.com/
[9]http://wordnet.princeton.edu/

Human Evaluation Second, the translated text went through three stages of human evaluation. *(1)* A monolingual group, consisting of 89 native and 194 non-native speakers of English recruited on MTurk judged a set of 3 randomly extracted sentences with respect to adequacy on an ordinal scale from 1 (None) to 5 (All Meaning) [27]. *(2)* A monolingual group, consisting of 283 participants (140 native and 143 non-native speakers), was asked to judge a randomly extracted sentence with respect to fluency on an ordinal scale form 1 (Incomprehensible) to 5 (Flawless English) [27]. *(3)* Finally, a bilingual group of 8 professional translators from the Swiss company 24translate[10] evaluated the translations by comparing each version of a translation to the German source text and rating them with respect to adequacy, fluency and grammar.

4.2.2 Results

Automatic Evaluation First, we compared the resulting quality of all 8 recombinations against the baseline. In direct comparison, two algorithms—CPxCP and CPxII—outperformed the baseline (0.29) by reaching a METEOR score of 0.38 and 0.36 respectively as shown in Table 2.

	METEOR	Precision	Recall	f1	fMean
CP x CP	0.389	0.76	0.71	0.74	0.71
CP x II	0.369	0.74	0.68	0.71	0.69
CP x DP	0.335	0.72	0.65	0.71	0.69
II x CP	0.290	0.68	0.64	0.68	0.66
II x II	0.290	0.68	0.64	0.66	0.65
II x DP	0.290	0.68	0.64	0.66	0.65
DP x CP	0.309	0.70	0.64	0.68	0.66
DP x II	0.290	0.68	0.63	0.66	0.64
DP x DP	0.298	0.69	0.65	0.66	0.64
Baseline	0.285	0.67	0.63	0.66	0.64

Table 2. Summary of METEOR evaluation

Furthermore, the resulting distribution of the automatic evaluation for each segment—a paragraph—shows that our translation algorithms approximated the reference literally (see Figure 12). Note that we regard the awful performance of most other recombinations not as a failure of our approach but as a desired result of a systematic exploration of the design space. Just like in biologic gene recombination many possible solutions are not viable. Nonetheless, an approach that explores all combination (or if computationally infeasible most combinations using some optimization approach) is more likely to uncover good solutions such as the CPxCP algorithm than one that tries to apply some kind of heuristic to immediately hone in on good ones.

Human Non-Professional Evaluation The second evaluation evaluated adequacy and fluency in a quantitative manner. The results are summarized in Table 3 and showed in Figure 13.
The 283 human non-professional evaluators rated the crowd-based translations in respect to adequacy and fluency on average as 3.16 and 3.37 on the ordinal scale from 1 (Incomprehensible) to 5 (Flawless English). In comparison, the professional reference translation reached on average 4.24 and 3.58 (see Figure 13). As such, the crowd-based algorithms outperform the baseline machine translation and are outperformed

[10]https://www.24translate.ch/

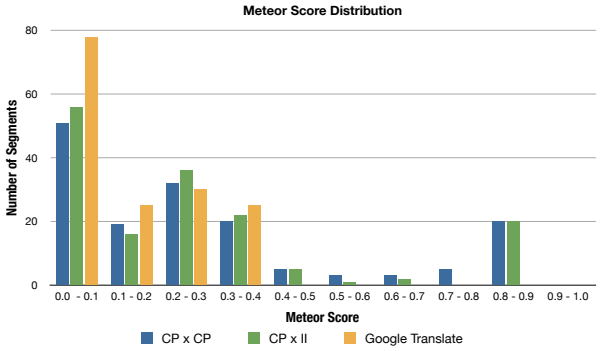

Figure 12. METEOR score distribution for each evaluated paragraph

by the reference translation. All differences are significant at the 95% level using the non-parametric Friedman test [13].

	ad_c	ad_p	fl_c	fl_p	gr_p
CP x CP	3.16 (σ=0.25)	3.5	3.37 (σ=0.22)	3	3
CP x II	3.14 (σ=0.22)	3	3.12 (σ=0.23)	3	3
Baseline	2.30 (σ=0.25)	1.5	2.18 (σ=0.22)	2	1
Reference	4.24 (σ=0.18)	5	3.58 (σ=0.24)	5	5

Table 3. Mean evaluation scores and standard deviation *(σ)* for the evaluation of adequacy *(ad)*, fluency *(fl)*, and grammar *(gr)* by professional translators *(p)* and crowd worker *(c)*

Human Professional Evaluation As shown in Table 3 and Figure 13, the 8 professional translators evaluated CPxCP as the best of all non-professional translation algorithms. As one can see CPxCP outperforms CPxII and the baseline in all three evaluation dimensions. While these results show that the resulting translations are far from perfect they still make useful translations available in a fraction of the time and cost of traditional solutions.

In particular, the analysis of the follow-up interviews with the professional translators and an in-depth analysis of the adequacy, fluency and grammar score distribution (see Figure 14)[11] that the differences in quality are mostly caused by a few challenges in the language structure which causes higher variance in the resulting translation.

For example, Translator-1 judges one of the translations as a *"Good solid translation that reflects exactly what the original says."*, whereas the pure machine translation failed which was expressed by Translator-2 *"Nonsensical. Tenses are the big issue here - mixed up in sentences and therefore makes the sense very difficult to understand."*. However, in some cases the CPxCP algorithm failed totally. As Translator-2 expressed by *"Slightly more meaning came across than in variant 1 [the baseline translation] but the grammar still poor."*

Using the professional translators reviews, we were able to identify several types of problems occurred in our experiments:

[11]The question whether the Likert scale should be considered equidistant or ordinal is under debate in the social sciences. Here, we interconnected the data points, for illustration purposes only without trying to take a stance on this question.

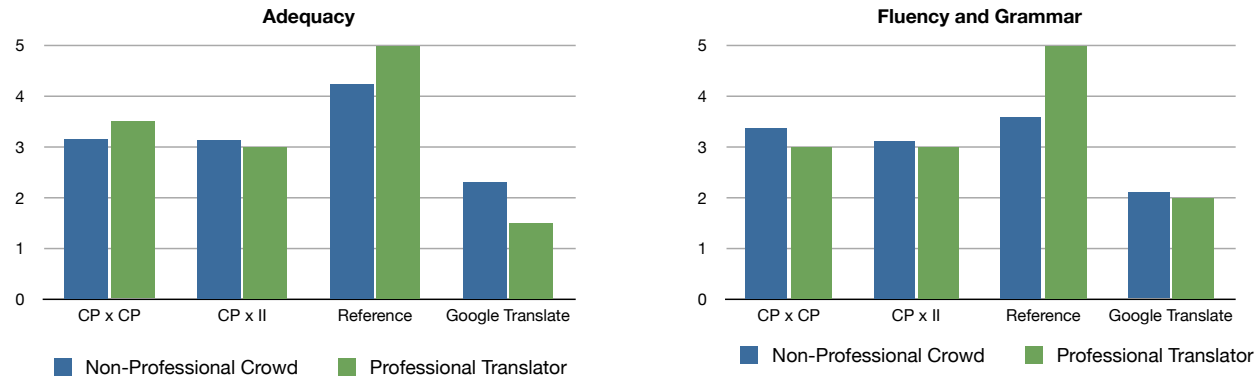

Figure 13. Mean evaluation scores for the evaluation of adequacy, fluency and grammar by 283 human non-professional evaluators and 8 professional translators

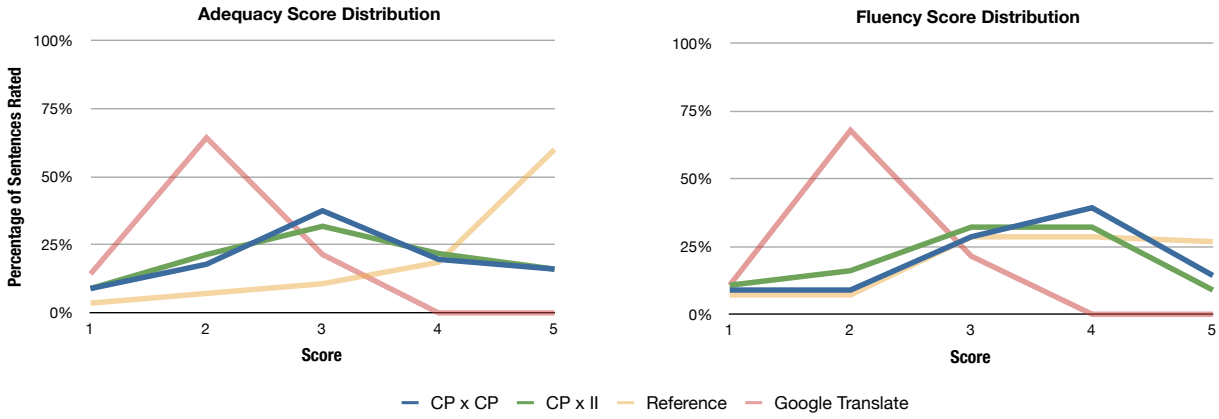

Figure 14. Proportional score distribution per paragraph for the different translation programs in regard to adequacy and fluency.

1. Word order and punctuation often leads to problems, when the word order provided by the machine translation reflects the morphology of the German language. Translator-5 elucidates this in detail: *"[...] reflects the German original: 'we clearly were in favor, Doha to continue'. Adverbs come after the verb 'to be' in English. So it doesn't sound very natural."*

2. Some translations struggle in using appropriate tenses as expressed by Translator-1 *"This would be fine except for two places that incorrectly use a relative clause with 'which' that render the sentences fragments rather than complete ideas and incorrect tense 'decline' should be 'declined'. A reader could still understand it, though, which is why I rated fluency higher than grammar in this case."* and Translator-2 *"Tenses an issue here. The German perfect tense is a past simple tense in English"*.

3. Finally, in very few instances problems where observed that should only occur when non-native speakers or machines are editing a translation. Translator-2 expressed this case as *"[...] and this [the problem] stands out as having been done by a non-native speaker or a machine."*.

We subsequently found that installing text-improvement "subroutines" in the program to address these specific challenges can significantly improve the results by still holding the throughput time and costs low. An empirical evaluation of these subroutines is forthcoming.

Performance Metrics On average, an article-translation was completed within 24 minutes for CPxCP or 35 minutes for CPxII. Considering work time only, a crowd worker spent on average (median) 113 seconds (1st Quartile (Q1) = 80 seconds, 3rd Quartile (Q3) = 250 seconds) on a task for CPxCP. In CPxII a crowd worker spent in average (median) of 115 seconds (Q1 = 78 seconds, Q3 = 188 seconds) on tasks. In terms of cost the translation of a sentence cost 0.09$ with CPxCP and 0.12$ with CPxII.

5. DISCUSSION, FINDINGS, AND LIMITATIONS

Our evaluation entails a number of interesting findings.

First, as the translation programs illustrate *CrowdLang* lends itself to the simple exploration of a large design space of possible program alternatives. Whilst we cannot provide empirical proof that this feature generalizes to a large number of other approaches it does, however, indicate that

a systematic exploration of the design space of possible human computation programs based on known and novel patterns may help to find good solutions. As a consequence, this technique promises to help the transition from an era of "Wizard of Oz techniques," where good functioning programs are the results of lengthy trial-and-error processes, to a more engineering-oriented era – a drawback of current human computation approaches. A goal first postulated by Bernstein et al. [6].

Second, the empirical evaluation shows that it is indeed possible to significantly improve the quality of generated translations employing monolingual crowd workers at astonishing speeds. Whilst the translations are far from perfect they make useful translations available in a fraction of the time and cost of traditional solutions. We are confident that the incorporation of further text improvement "subroutines" in the program, such as the use of bilingual crowd workers for the most complex German sentence structures only, will significantly improve the result.

Third and most astonishing, our adaptation of the six-sigma rule to human computation allows us to run the processes without any sophisticated pruning techniques. Specifically, we could forgo any use of "control questions" or qualification tests – a considerable saving in terms of effort. On the downside, however, our evaluation is limited in that a usage of such quality control measures may have lead to better results. We will have to investigate this question in the future.

Fourth, our pairing of the systematic exploration of the design space with the empirical evaluation helped us to find a novel human computation pattern CPxCP that we call *Staged Contest with Pruning*. This best performing pattern combined contests over several stages by pruning the intermediate results using the six-sigma rule and automatic comparison with the input to uncover cut-and-pastes. We will support the belief that this pattern will be highly useful in other applications with empirical evidence in a future study.

As a major limitation our programs were so far only evaluated in German to English translation tasks. To address these limitations we will need to evaluate the programs using a variety of existing machine translation tasks such as the EU parliament datasets, prose texts, as well as less common language pairs. In addition, we need to explore the sensitivity of our programs to different machine translation tools. In particular, we need to explore how the programs react to better or worse initial machine translations.

Last but not least, there might be limitations to the parallelization we employed. When translating longer texts such as a whole book some expressions may have to be translated consistently to the same word. Our approach would not guarantee the consistent usage. Hence, we may have to extend the approach with additional measures to ensure consistent translations.

6. CONCLUSION AND FUTURE WORK

In this paper we introduced *CrowdLang* – a general-purpose framework and programming language for interweaving human and machine computation. Using the practical task of text translation we illustrated that *CrowdLang* allows the "programming" of complex human computation tasks that entail non-trivial dependencies and the systematic exploration of the design space of possible solutions via the recombination of known human computation patterns.

Our empirical evaluation showed that some of the resulting programs generate "good" translations indicating that the combination of human and machine translation could provide a fruitful area of human computation. Finally, it unearthed a novel human computation pattern: "the Staged Contest with pruning".

We hope that *CrowdLang* will be used by others to implement their human computation programs, as it will allow them to easily try out and compare different solutions. Whilst *CrowdLang* is not the first human computation language it is definitely another step in the development of human computation programing language.

ACKNOWLEDGMENTS

We are grateful for the support by Sven Lendi and Luca Vidi from the language service provider 24translate (www.24translate.ch) which sponsored the evaluation. In addition, we are thankful for the research assistance by Patrick Leibundgut and David Oertle. We would also like to thank Tom Malone for his support to the ideas underlying this paper. Special thank to Nicolas Hoby, Minh Khoa Nguyen, Elaine Huang, Philipp Stutz, Cosmin Basca and Lorenz Fischer for valuable feedbacks

REFERENCES

1. Ahmad, S., Battle, A., Malkani, Z., and Kamvar, S. The jabberwocky programming environment for structured social computing. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, ACM (2011), 53–64.
2. Banerjee, S., and Lavie, A. Meteor: An automatic metric for mt evaluation with improved correlation with human judgments. *Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization* (2005), 65.
3. Bernstein, A. How can cooperative work tools support dynamic group process? bridging the specificity frontier. In *Proceedings of the 2000 ACM conference on Computer supported cooperative work*, ACM (2000), 279–288.
4. Bernstein, A., Klein, M., and Malone, T. The process recombinator: a tool for generating new business process ideas. In *Proceedings of the 20th international conference on Information Systems*, Association for Information Systems (1999), 178–192.
5. Bernstein, A., Klein, M., and Malone, T. Programming the global brain. *Communications of the ACM 55*, 5 (2012), 1–4.
6. Bernstein, M., Little, G., Miller, R., Hartmann, B., Ackerman, M., Karger, D., Crowell, D., and Panovich,

K. Soylent: a word processor with a crowd inside. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, ACM (2010), 313–322.

7. Brabham, D. Crowdsourcing as a model for problem solving. *Convergence: The International Journal of Research into New Media Technologies 14*, 1 (2008), 75.
8. Chase, R., Aquilano, N., and Jacobs, F. *Operations management for competitive advantage*. McGraw-Hill/Irwin New York, 2006.
9. Chen, Y., Liem, B., and Zhang, H. An iterative dual pathway structure for speech-to-text transcription. In *Human Computation: Papers from the AAAI Workshop (WS-11-11). San Francisco, CA, August* (2011).
10. Dean, J., and Ghemawat, S. Mapreduce: Simplified data processing on large clusters. *Communications of the ACM 51*, 1 (2008), 107–113.
11. Franklin, M., Kossmann, D., Kraska, T., Ramesh, S., and Xin, R. Crowddb: answering queries with crowdsourcing. *Proceedings of SIGMOD 2011* (2011), 61–72.
12. Howe, J. The rise of crowdsourcing. *Wired magazine 14*, 14 (2006), 1–5.
13. Iman, R., and Davenport, J. Approximations of the critical region of the friedman statistic. Tech. rep., Sandia Labs., Albuquerque, NM (USA); Texas Tech Univ., Lubbock (USA), 1979.
14. Kittur, A., Smus, B., Khamkar, S., and Kraut, R. Crowdforge: Crowdsourcing complex work. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, ACM (2011), 43–52.
15. Kulkarni, A., Can, M., and Hartmann, B. Turkomatic: automatic recursive task and workflow design for mechanical turk. In *Proceedings of the 2011 annual conference extended abstracts on Human factors in computing systems*, ACM (2011), 2053–2058.
16. Kulkarni, A., Can, M., Hartmann, B., et al. Collaboratively crowdsourcing workflows with turkomatic. In *Proc. CSCW* (2012).
17. Law, E., and Ahn, L. Human computation. *Synthesis Lectures on Artificial Intelligence and Machine Learning 5*, 3 (2011), 1–121.
18. Law, E., and Zhang, H. Towards large-scale collaborative planning: Answering high-level search queries using human computation. In *Twenty-Fifth AAAI Conference on Artificial Intelligence* (2011).
19. Little, G. *Programming with Human Computation*. PhD thesis, Massachusetts Institute of Technology, 2011.
20. Little, G., Chilton, L., Goldman, M., and Miller, R. Exploring iterative and parallel human computation processes. In *Proceedings of the ACM SIGKDD workshop on human computation*, ACM (2010), 68–76.
21. Little, G., Chilton, L., Goldman, M., and Miller, R. Turkit: human computation algorithms on mechanical turk. In *Proceedings of the 23nd annual ACM symposium on User interface software and technology*, ACM (2010), 57–66.
22. Malone, T., and Crowston, K. The interdisciplinary study of coordination. *ACM Computing Surveys (CSUR) 26*, 1 (1994), 87–119.
23. Malone, T., Laubacher, R., and Dellarocas, C. The collective intelligence genome. *MIT Sloan Management Review 51*, 3 (2010), 21–31.
24. Malone, T., Laubacher, R., and Johns, T. General management: The age of hyperspecialization. *Harvard Business Review 89*, 7-8 (2011), 56–65.
25. Minder, P., and Bernstein, A. In *Workshops at the Twenty-Fifth AAAI Conference on Artificial Intelligence* (2011).
26. Noronha, J., Hysen, E., Zhang, H., and Gajos, K. Platemate: crowdsourcing nutritional analysis from food photographs. In *Proc. of the 24th annual ACM symposium on User interface software and technology*, ACM (2011), 1–12.
27. Papineni, K., Roukos, S., Ward, T., and Zhu, W. Bleu: a method for automatic evaluation of machine translation, 2002.
28. Quinn, A., and Bederson, B. Human computation: a survey and taxonomy of a growing field. In *Proceedings of the 2011 annual conference on Human factors in computing systems*, ACM (2011), 1403–1412.
29. Von Ahn, L. Human computation. In *Design Automation Conference, 2009. DAC'09. 46th ACM/IEEE*, IEEE (2009), 418–419.
30. Wang, F., Carley, K., Zeng, D., and Mao, W. Social computing: From social informatics to social intelligence. *Intelligent Systems, IEEE 22*, 2 (2007), 79–83.
31. Young, H. An axiomatization of borda's rule. *Journal of Economic Theory 9*, 1 (1974), 43–52.
32. Zhang, H., Horvitz, E., Miller, R., and Parkes, D. Crowdsourcing general computation. In *ACM CHI 2011 Workshop on Crowdsourcing and Human Computation* (2011).
33. Zhang, H., Law, E., Miller, R., Gajos, K., Parkes, D., and Horvitz, E. Human computation tasks with global constraints. CHI (2012).