Climbing Mont Blanc – A Case Study in Challenging the Most Eager Students in a Large Programming Class

Lasse Natvig*, Magnus Själander, and Magnus Lie Hetland

June 2018

Abstract

Climbing Mont Blanc (CMB) is a system developed by students for students. Its aim is to inspire students to learn more about the energy efficiency of programs. CMB facilitates the evaluation of programs executed on modern heterogeneous multi-core processors such as the Exynos Octa chips used in, e.g., the Samsung Galaxy S5 and newer mobile devices. CMB evaluates execution time, energy efficiency, and a user-specified quality-metric for problems where this has been defined. Student submissions can be ranked according to these different metrics, and the system can be used to host programming competitions. By arranging competitions we encourage the participants to study and improve their own solutions in order to climb the scoreboard. This provides a strong incentive for the students to get a better understanding of energy efficient programming through exploring and testing different solutions.

In the last two years we have organized a fully optional competition as part of the course TDT4102 Procedural and Object-Oriented Programming at NTNU to challenge the most eager students in energy efficient programming. Seven new problems were developed for the CMB Challenge 2018, and 22 students took part. A total of 732 programs were submitted and 947 tests were performed with results collected and reported. It was a close race between the top three contestants and running the competition gave lots of useful feedback to the CMB-project regarding technical solutions and how to design problems. It also gave new insight among students and the course teacher about elements of energy efficient C++ coding.

We briefly present the CMB system, the problem sets developed for the competitions, and the main insights gained so far. CMB is a long-term project. The system is still under development, and is available for other interested teachers and researchers.

Keywords: *energy efficiency, programming competition, algorithms, combinatorial optimization.*

*The authors are affiliated with the Computing group in the Department of Computer Science at NTNU in Trondheim, and the first author can be contacted at Lasse@computer.org.

1 Introduction

Climbing Mont Blanc (CMB) is an open online judge for training in energy efficient programming on a processor typical for contemporary mobile devices. The spring 2018 we organized a competition in energy-efficient programming as an optional activity in the course TDT4102 Procedure and Object-Oriented Programming at NTNU. The main motivation for the *CMB Challenge 2018* was twofold. First, we wanted to inspire and challenge the best or most eager students in the class. Second, we wanted more user testing of the Climbing Mont Blanc system.

The first CMB version was available in 2015 [FS15], and it has been further developed by several Master's students (See Section 2). To the best of our knowledge, CMB is the only on-line programming judge with energy efficiency as a metric. CMB is a distributed software system using many different software technologies and developed by students coming, and leaving. Reliability and maintainability by the TDT4102 course teacher has been a challenge, and the progress in functionality, quality, and reliability has been slow but steady. CMB has so far received more than 6000 submitted programs and 11 300 executions have been measured during the last three years. It accepts C and C++ programs, with support for OpenCL 1.1, OpenMP 4.0, and Pthreads.

The course has about 800 students coming from more than 50 different study programs. Most of them learn Python or Matlab in their first semester and take TDT4102 in their second or fourth semester. Less than 10 % of the students are computer science (CS) students since the CS students at NTNU Gløshaugen learn Java and not C++. The course has 12 extensive exercises, and a student needs an approval on eight of these by a teaching assistant to be allowed to take the final four hour written exam. With more than 50 teaching assistants there are 60 hours of supervised lab activity per week. Furthermore, we have conventional lectures, exercise lectures, and more informal lectures given by teaching assistants. All the main activities in the course are aimed at the main bulk of the class. At the end of the semester we offer extra supervision to those students that are in danger of failing the course due to too few approved exercises. The set of exercises gives a labour-intensive course, and it was not expected that a large fraction of the students wanted even *more* programming tasks. The purpose of the competition was to give extra inspiration and learning *for the best or most eager students*.

A similar but smaller competition was organized in the spring of 2017 where we used problems already available on CMB. We experienced a need for some simpler problems, and decided to develop a new set for the 2018 competition. The development of these new problems was motivated by two research questions:

Research question 1 (RQ1) How are different kinds of problem formulations for CMB received by the students?

Research question 2 (RQ2) How does the use of CMB in TDT4102 affect motivation in the course and further studies?

The paper is organized as follows. In Section 2 we briefly introduce the CMB

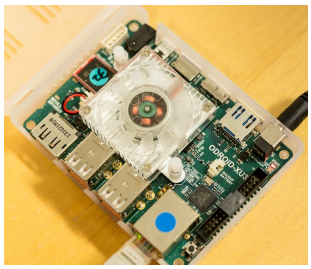


Figure 1: Odroid XU3 used to execute uploaded CMB programs. It is available at `https://climb.idi.ntnu.no` through a graphical user interface. It contains a modern heterogeneous processor typical for recent smart phones. The user select from a menu of programming problems and submissions are ranked according to execution time or energy efficiency.

system. Section 3 presents the competition with focus on the programming problems. In Section 4 we discuss aspects of preparing the problems, giving advice to students during the 4-week competition and compiling a fair list of results. Feedback from the user questionnaire is presented in Section 5, related work is briefly covered in Section 6, and we conclude and sketch further work in Section 7.

2 A Brief Presentation of Climbing Mont Blanc

The system uses an Odroid-XU3 development board from Hardkernel [Odr] with a Samsung Exynos 5 Octa SoC (System-on-Chip) [Sam]. The XU3 has integrated power sensors facilitating energy-efficiency measurements. The SoC contains four ARM Cortex-A15 cores at 2 GHz and four ARM Cortex-A7 cores at 1.3 GHz, an ARM Mali-T628 GPU with six cores, and 2 GB of RAM at 933 MHz. Thus, it can be called a three-way heterogeneous computing platform with 14 cores.

We are currently working on extending the CMB system with an alternative newer execution platform – a HiKey960 board [HiK]. It has a more powerful but similar processor architecture from Huawei, with 16 cores. The most recent Exynos SoC is the Exynos 9810 used in the Samsung Galaxy S9/S9+ mobile phones, which arrived on the market in March 2018 [Sam]. This chip has 28 cores using the same three-way heterogeneity as the Exonys in the Odroid-XU3. There is little focus at IDI on teaching students how to program these mass-market processors efficiently. It is therefore an *overarching goal of the CMB project to motivate more students in training for energy efficient programming of these highly challenging platforms*. The CMB project is part of the strategic research area Energy Efficient Computing Systems (EECS) at NTNU [EEC].

To implement software applications on these architectures with state-of-the-art energy efficiency requires programming skills far beyond the course scope of TDT4102, which is at a basic level. CMB currently accepts C and C++ programs, the GPU can be programmed using OpenCL v1.1, and the parallel nature of the processor chip can be exploited using OpenMP 4.0 or Pthreads. However, in TDT4102 we do not expect the students to dive into parallel programming. The focus is solely on C++ programming and uni-processor execution.

CMB accepts programs submitted using a web-based graphical user interface, and they are evaluated with respect to time, energy used, and energy-delay product (EDP). EDP is one of the most common metrics used in research on energy efficient computing systems, often called green computing. A small and varied set of problems are available, and the system is open for use by any interested programmer. Other online programming judges exist (See Section 6), but we are not aware of any similar system reporting energy-efficiency.

More information about the project can be found at its official webpage [Nat]. A compact introduction to the CMB system including a technical overview is found in an earlier paper [Nat+]. The first version of CMB was developed by Torbjørn Follan and Simen Støa and is documented in their Master's thesis [FS15]. Their work was further improved by Sindre Magnussen, who made significant improvements to the user interface [Mag16]. In the spring of 2017, Fredrik Pe Ingebrigtsen worked to improve the reliability of the system [Ing17]. He also implemented a profiling option, which gives detailed feedback to the user, including performance counter values and flame-graphs [Gre16]. Figure 1 shows the back-end of the CMB system in action. It is connected to a server running the CMB database, which again is connected to the front-end that presents the user interface.

3 The TDT4102 CMB Challenge 2018

In addition to providing a motivating spectrum of challenges for the students, we wanted to investigate whether the students prefer standalone problems or a series of problems that build on each other. We used both short formulations of toy example problems on numbers, and more application-like problems. In addition to CMB, the students have been directed to Kattis [Ens+11] for extra programming training. There, many of the problems have a slightly humorous style of formulation with flavor text added to the problem description. In line with our research questions (RQ1, Section 1) we did a user-survey with a questionnaire to learn about their preferences in this respect (See Section 5).

First we present an overview of the seven problems. Thereafter, the series of related problems, with short names OGAP-1/2/3, are discussed in more detail. They are related to a more complex classical combinatorial problem. This problem gives room for several very different solution algorithms, and we hope the exposure to some of these will motivate students to further studies in algorithms (RQ2).

3.1 The Seven Programming Problems – Overview

The programming problems developed for the TDT4102 CMB Challenge 2018 are summarized in Table 1. The first column gives the short name used in the further discussion. *Type* describes the problem type in line with the discussion above. *Submissions* is the number of programs uploaded by contestants during competition, and *runs* is the number of programs executed and measured. In the

Table 1: Summary of CMB Challenge 2018 problems, submissions and runs.

Name	Type	Submissions	Runs	Description
Crunchy	Number	297	416	Sum of numbers
Sigma Unique	Number	167	220	Sum of unique numbers
Top Forty	Number	62	75	Most frequent numbers
OGAP-1	Series	63	72	Assign seats by student
OGAP-2	Series	55	69	Assign seats by group
OGAP-3	Series	66	72	Optimize seat assignment
Short&Fast	Applic	22	23	Classical shortest path

competition, the contestants were given one point for every solved problem, three extra points for the fastest solution, two for 2nd fastest and one for 3rd place. Similarly, three, two or one points for the lowest EDP-values. For the OGAP-3-problem we used a different scoring system to focus more on its optimization aspect, as is described below.

The three first problems are simple, relatively independent, standalone problems dealing with numbers, more like toy examples. They are all small and easy, but have a slight increase in difficulty. The last problem is a standalone application-oriented problem, and it is estimated to be difficult for students in this course since it is far beyond its curriculum. However, since it is a classical problem, the students are expected to find useful algorithms such as, e.g., Dijkstra's shortest path algorithm.

OGAP is short for Optimal Group Assignment Problem and is motivated and inspired by a practical task involved in the start-up phase of the TDT4102 course. The task is to distribute the 780 students among 46 different lab groups, and the challenge is to ensure that all students get a group that fits well into their time schedule, as well as achieving a relatively even distribution between the groups.

For the problems in the competition, we first described a very simple algorithm for finding a solution in OGAP-1, then followed by a different but still simple algorithm in OGAP-2. OGAP-3 is also the same problem but made more complex by asking for a best possible solution. The idea was that this *series of problems* should bring the students step-by-step into an interesting application that could motivate further studies. We will now briefly introduce the four standalone problems before we discuss the OGAP-problems to a greater extent in the next subsection.

Crunchy is the simplest number-crunching task we could come up with. The task is to read a large number (N) of integers from standard input, and print out the sum of the numbers. All the numbers are positive and smaller than 100, and $0 < N < 20\,000\,000$.

Sigma Unique also deals with a large number of integers. Here the students are asked to calculate the sum of all integers that are unique. A natural

way of solving this for students in TDT4102, which focus on modern C++ with STL is to use the `set`-template in the library. While the numbers are read, new numbers are inserted in a set and added to the sum while numbers already in the set are simply ignored.

Top Forty is solved by counting the frequency of numbers read, and reporting the top 40 most frequent numbers in the input. Again, there was a "modern C++ pedagogical motivation"; an STL `map` combined with a priority queue from STL `queue` yields a very simple and efficient solution.

Short&Fast is the classical algorithmic problem called *single source shortest path*. It has been available in CMB for almost three years. The best student in the CMB Challenge 2018 managed, to our surprise, to solve the problem about 50 times faster than any previous solution. Further studies will be done to investigate the reasons for this excellent performance.

3.2 The Optimal Group Assignment Problem

As introduced above, the *Optimal Group Assignment Problem* (OGAP) is motivated and inspired by the practical task of assigning students to lab-groups.

The **OGAP-1 problem** was specified as follows: The first input line contains the three integer variables N, G and S, where $0 < N \leq 10\,000$, $0 < G \leq 1000$, and $0 < S \leq 100$. N is the number of students, G is the number of groups, and S is the number of seats in every group. This is followed by N lines of text, where the first is the *priority list of selected groups* for student no 1. On the next line is the priorities of student no 2 and so on. Both groups and students are numbered starting with 1. Every student has selected every group in some priority order, and filling every group to its maximum will give all students one seat. This implies that $G \times S = N$.

In the example shown in the left part of Figure 2, student no 1 gives priority value 1 to group 1, priority 2 to group 4, priority 3 to group 3 and priority 4 to group 2. Student no 2 has the same priorities. Student 3 has group 4 as priority 3 etc. In this specific variant of the OGAP problem you are asked to assign groups to students following the algorithm called OGAP-1-Solve shown in the right part of Figure 2. The algorithm assigns the best possible group to every student, in the same order that the students have in the input list. For our input example, the program should produce the group assignment (output) as shown in the centre of the figure. The last line should give the value called *Sum priorities*, which is the sum, over all students, of the priority in the list of selected groups for the group assigned to the student. In our case it will be $(1 + 1) + (2 + 2) + (3 + 3) + (3 + 3) = 18$. The students were told that the big problem instance hidden in the CMB system was much larger, but they did not get the actual size (8000 students and 400 groups with 20 seats each).

The **OGAP-2 problem** is the same as OGAP-1, but with the difference that the students were asked to use a quite different but equally simple algorithm to calculate an assignment of seats. Seat assignments are done group by group, by picking those students that have a given group as highest priority (lowest

```
8 4 2
1 4 3 2
1 4 3 2
1 2 4 3
1 2 4 3
2 1 4 3
2 1 4 3
1 2 3 4
1 2 3 4
```

```
Group 1 : 1 2
Group 2 : 3 4
Group 3 : 7 8
Group 4 : 5 6
Sum priorities 18
```

OGAP-1-Solve

```
1  for every student in numbered order
2      for every selection in priority order
3          if the selection is a group with free seats
4              assign the student to that group
```

Figure 2: Input, expected output and algorithm for the OGAP-1 problem.

```
Group 1 : 1 2
Group 2 : 5 6
Group 3 : 7 8
Group 4 : 3 4
Sum priorities 16
```

OGAP-2-Solve

```
1  for every group in numbered order
2      for priority p = 1..G
3          Fill the group with non-assigned students
4              having this group as priority p
5              in the student order until the group is full
```

Figure 3: Expected output and algorithm for the OGAP-2 problem. The input example is the same as for OGAP-1.

priority value) first, then those with second-highest etc. The algorithm and its produced output for the same input example as for OGAP-1 is shown in Figure 3. With this input example, OGAP-2 gives a better result for the sum of priorities (16).

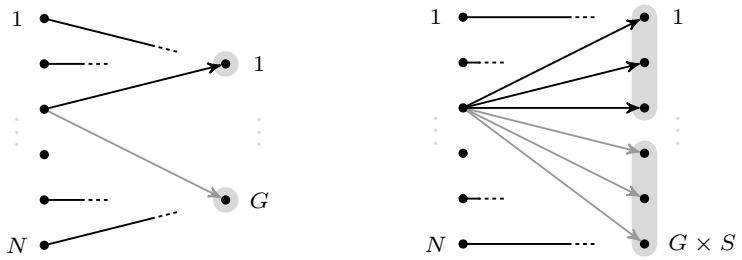
The **OGAP-3 problem** gives a more challenging task. The problem is the same as OGAP-1&2, but we defined two other ways to score points that asked for much smarter approaches than those used in OGAP-1&2. CMB gives the problem creator the possibility to define a problem specific *goodness-metric* that can be used to rank submitted programs. In the context of OGAP it is natural to compete in achieving the lowest value for sum priorities (hereafter called *SP*). We awarded six, four, and two points for the three lowest values for *SP*, regardless of energy and time used. We also defined a combined metric as $EDP \times 10 \times SP$, where lowest value is best, and six, four, and two points were awarded to the three best solutions.

The OGAP-3 problem opens the door to a variety of different solution algorithms, and we mentioned a few of these in the presentation given to the students after the competition – to motivate further studies in algorithms. Here we sketch a few:

(**By luck**) Assign students to groups in student-order as in OGAP-1, but with the order of the students randomized. Repeated runs of this simple algorithm will produce different assignments and different values of *SP*, demonstrating that the result to a large extent relies on luck.

(**Brute force**) A slight improvement is to repeat the "By luck method" with as many iterations as CMB allows[1] and print the best solution found. This is a very energy-*inefficient* method but illustrates the principle of "brute force."

[1] CMB runs submitted programs for a maximum of 60 s before they are timed out.



(a) Assigning to groups (b) Assigning to seats

Figure 4: The reduction from OGAP to the assignment problem. Each group is split into S seats, inheriting original edge weights. Though the new instance size now depends on S, it is polynomial as a function of the original, because $G \times S = N$.

(**Swap**) A much better method is to start with some assignment from OGAP-1 or -2, and then try to swap two and two randomly selected students and see if it gives a better assignment. This can be repeated as long as there is progress towards a lower SP.

(**Simulated annealing**) Define a mutation operator, such as the swap strategy above, but apply it *tentatively*. If it results in an improvement, accept the new configuration; otherwise, accept it with a probability that decreases over time.

(**Flow**) Probably the most natural formalization of the problem is as a minimum-cost flow problem with supply and demand, which is solvable using standard algorithms [AMO93]. Each student node would have a supply of 1, each group node would have a demand of S, and the student–group edges would have a cost equal to the priority. Empirically, the fastest algorithm for this problem is the network simplex algorithm, though this can be very complex to implement [Orl97]. A reasonably straightforward solution is the successive shortest path algorithm, which in this case would have a running time of $\mathrm{O}(N^4)$ [AMO08].

(**Assignment**) An alternative is to reduce the problem to the pure assignment problem, as shown in Figure 4, and then using, for example, the Kuhn–Munkres algorithm. A straightforward implementation would have the same running time, though this could be improved to $\mathrm{O}(N^3)$ [BDM12, p. 85].

Only two of the students found a solution for the OGAP-3 problem with a significantly better value for SP than what is found by our two naïve algorithms from OGAP-1 and -2. However, the winner of the competition submitted a program giving $SP = 9238$, almost 10 % better than the second best (10 088). The winner had five times better EDP than the simple OGAP-1/2 algorithms, and a value for the combined metric that was an order of magnitude better than the second best. His strategy can informally be described as follows:

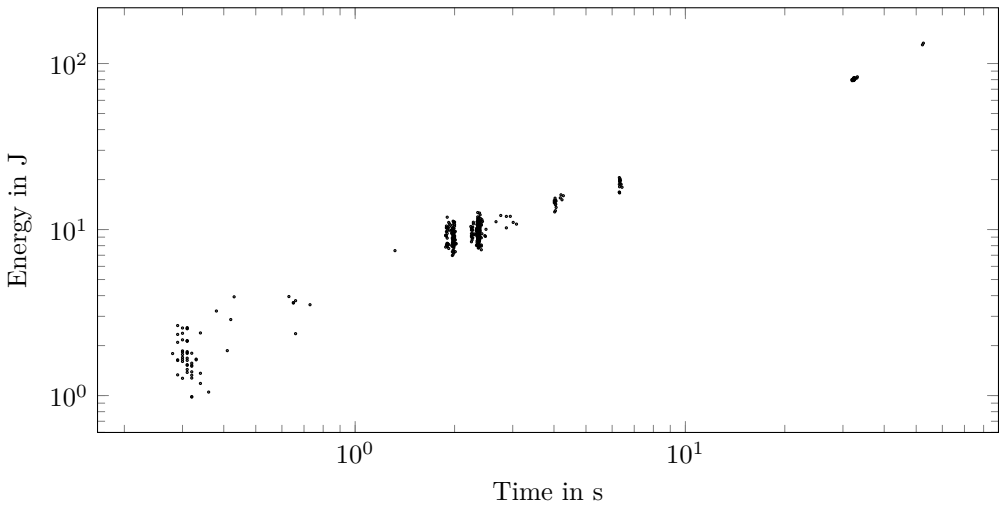


Figure 5: Performance and energy of the 416 runs for the Crunchy problem.

> *Winning solution:* Read the students priorities and assign each of them to their most wanted group. This most probably gives a "super-optimal" assignment where some groups have more students than there are seats. This can be fixed in a "greedy" manner going through all groups with too many students. First, the program tries to move students from their overfull 1st priority group to their 2nd (or 3rd or ...) until there is a free seat. This is done per priority level for all students that must be moved. Students are only moved if the new group is not full.

We plan to compare the performance of the winning solution with some of the more well-known algorithms in a future study.

4 Preparing, running, and finishing the competition

Figure 5 illustrates several practical matters related to the preparation, running, and finishing of the competition. A simple tool processed database-dumps from CMB for each of the seven problems during the competition and was used to get a quick overview of the status. The figure plots the performance of the 416 measured executions of programs solving the Crunchy problem. The plot uses a log-log scale and shows a strong correlation between time and energy. This is as expected since almost all the submissions are single-threaded program. More advanced programming with multithreading and process-binding to the less power-hungry cores and the GPU-cores on the XU3 would probably give an even more interesting picture.

In the upper right corner of the plot we have programs using 30–50 s. We designed the simplest problems to have an execution time of about 30 s for those students that did *not* read the HowTo-page at the CMB website. The HowTo-page gives brief instructions on how to upload programs, and recommends that students turn off synchronization of the standard C++ streams. This enables the C++ standard streams to buffer their I/O independently, which is considerably faster in some cases[2].

The two big clusters of dots in the center of the figure are those programs that had turned off the synchronization. Our own very simple solution was in this area. Midway in the competition there were several contestants that still were in the upper right corner, while already after a few days we got submissions that were surprisingly fast in the lower left corner. We were worried that we might lose the group in the upper right corner to the very hard competition, and so we reminded all participants about reading the HowTo-page.

In the centre of the figure are two clusters of runs, one with an execution time of about 2.35 s, typical for very simple textbook solutions, i.e., a loop reading from standard input to an integer variable and forming the sum. The cluster around 2.00 s are runs where the students pre-increment rather than post-increment the loop-counter, and have chosen more specific data types like `Uint32` or `Uint16` from the newer C++ standards.

Note that the energy readouts are less precise than the time measurements. This is seen as more variation along the y axis than the x axis, since many of the dots come from different runs of the same program. Follan and Støa reduced the deviation in time and energy-readouts from the Odroid XU3 in three ways [FS15]: (*i*) Cooling down (if necessary) and then warming up the processor to a fixed temperature before starting the measured execution. In earlier work, we have found that changing temperature from 30 to 70 degrees Celsius can give as much as 10–12 % difference in processor energy [CN13]. (*ii*) Clearing the cache before every run. (*iii*) Disabling the display manager (`lightDM`, in Ubuntu), as the XU3 does not have a display. The combined effect of these three techniques reduced the relative standard deviation from 2.5 % to 0.15 % for the execution time on a problem with typical run time of about 40 s. However, we have typically seen larger variations in the *energy* readouts in general, and especially for very short runs. In the successor board Odroid XU4 the energy monitors are *not* present, and the CMB project is continuously looking for improvements in this area.

Due to the above mentioned challenges in the precision of measurements the announced scoring system described the possibility that points could be shared between two or three students, if the difference in a competition metric was smaller than its expected precision. In addition, to reduce the relative standard deviation, we wanted to have an execution time of 20–30 s even for the simplest problem, Crunchy. That implied a very large problem instance, described by an input-file using 33.3 MB. A consequence was that fast I/O became much more important than we had planned, and it gave some unexpected extra work.

The solutions in the lower left corner are in the area 0.3–0.4 s, and far better

[2] `http://en.cppreference.com/w/cpp/io/ios_base/sync_with_stdio`

than we expected. Quite early one of the contestants found out that reading the big input file as a raw byte stream and take the responsibility of translating to integers instead of letting the standard library do this really paid off. Using `getchar_unlocked()`[3] reduced the execution time even more. Later, two other students found similar solutions, and this paid off to a large extent for all the three number problems (See Table 1). Besides giving more focus on I/O than we wanted, it also made compiling the final list of results harder. To compare these very fast solutions in a fair way required many reruns since they were in the area of high deviation.

5 Feedback Questionnaire

The competition was held during the four weeks from March 8th to April 6th in 2018. In the following week some simple solutions to the problems and the final lists of results were presented to the students. This was followed by a short questionnaire among those 22 students that took part of the challenge. We got 17 answers. The questionnaire had ten questions in total. In the following, summarize and visualize the responses to the four questions that are most relevant for the research questions presented in the introduction.

Q2 "In the CMB challenge we had three different types of problems. (a) Simple, standalone number-problems like Crunchy, Sigma Unique and Top Forty, (b) a series of three problems related to the application "assign students to groups" (OGAP-1/2/3) and (c) a classical algorithmic problem (Shortest path). Which of the three types did you like most when you ignore differences in difficulty?"

a	b	c
58.8 %	23.5 %	17.6 %

Q4 "It is a tradition in programming competitions that problems are formulated with a good deal of superfluous text and often with a humorous touch. (See, e.g., Kattis), here called type (a). Another type (b) are more short and concise problems, often without any specific application (so called toy examples) specifying only what you have to know (e.g., our Sigma Unique). A third type (c) are real problems with a description of an application, but not more text than necessary (e.g., OGAP or Short&Fast). Which type do you like best?"

a	b	c
52.9 %	11.8 %	35.3 %

Q6 "How would you rate the user-friendliness of CMB?" (a) Very good, (b) Good, (c) Medium, (d) Poor (e) Mediocre.

[3] This is a faster version of `getchar()` but it is deprecated since it is not thread safe. However, since TDT4102 only deals with single-threaded programs it should be allowed in this context.

11.8 % 76.5 % 11.8 %

Q9 "To what extent do you think getting feedback on execution time and energy efficiency give you increased understanding of your own code?" (a) To a very large extent, (b) To a large extent (c) To some extent (d) Not at all (e) It is only confusing.

23.5 % 23.5 % 47.1 % 5.9 %

With respect to RQ1, the answers to Q2 clearly show that most of the competing students prefer the standalone number problems, and the response to Q4 clearly suggests formulating at least some of them in the more lengthy and humorous "Kattis-style." We can also see that some of the students liked the OGAP-series of problems and application-like problems, whereas it does not seem to be important that the problems are short and concise.

The response about CMB user friendliness (Q6) was surprisingly good – the fact that the competition was optional may have affected these results. Note that no one answered alternative (d) or (e). The questionnaire also had a fill-in form and we got concrete feedback on some issues that should be improved; many of these are technical issues with the GUI we are aware of.

With respect to RQ2, the answers to Q9 are very positive and give us strong motivation for continued work on CMB with new competitions. In a related fill-in form, several students wrote that they would like to read more literature about the energy-efficiency of programs and many of them expressed a desire for continued study within this area. Most of the students wanted to get updates from the CMB-project also *after* having completed TDT4102.

6 Related Work

The CMB project is motivated by the large and increasing activity on online programming judges such as UVa Online Judge [RML08], PKU Judge Online [Pek], and Kattis [Ens+11]. A good glimpse into the ongoing activity on UVa Online is available at uHunt[4] that typically shows 4–5 submissions every minute!.

The NTNU course TDT4120 Algorithms and Data Structures has used a similar system for their exercises for over 15 years, which was also built from scratch.[5] That system has been very well received, but is getting old and increasingly unmaintainable, and has no support for energy measurements. The course TDT4120 was an initial inspiration for the long term work on the CMB project.

CMB Master's theses [FS15; Mag16; Ing17] give numerous references to online judges, but we are not aware of any other online system reporting energy

[4] `http://uhunt.onlinejudge.org/`
[5] `https://tdt4120.idi.ntnu.no`

efficiency. A literature review is given by Ihantola et al. [Iha+10]. The closest work we have found is the Green Coding Contest that was organized by Intel in 2014 [Int14].

7 Concluding Remarks

The CMB Challenge 2018 has been a boost for the CMB project giving lots of feedback to the system as well as contact with many interested students. However, a few weaknesses in the contest and our study have been identified.

The three number-problems were in a way a series of problems, and this reduces the value of the feedback from the questionnaire on the students preferences with respect to problems (standalone vs. series). Next year, we will design problems to be more independent and formulate one or two of them in the popular "Kattis-style."

Perhaps more importantly, as explained in Section 3, an unfortunate aspect of the competition was that too many of the problems were too I/O-bound. Fast reading of the problem input set from file took too much of the focus. For our next competition, this will be solved by producing problem instances and verifying solutions programmatically, based on secret seeds for the pseudorandom generator.

A challenge inherent in online judges used for competitions is that feedback during program development often is rather restricted, an issue also mentioned by some of the students in the questionnaire. To some extent it is "blind programming" without a debugger, and with reduced error messages. We tried to partly reduce this drawback with an online forum where contestants could discuss the various problems, also giving a fair way to help struggling students, since the forum, including any answers given, is available to all.

One student solved many of the problems during the first days of the competition with outstanding performance, and this may have demotivated other students. For future competitions, an idea could be to show only how many students had solved problems during the first two weeks, and unveil the performance of the different solutions only during the last part of the competition.

In addition to a new and improved competition next spring, we will continue on improving the CMB project: We will extend the documentation with general notes on "how to make your C++ programs more energy efficient,", i.e., a kind of "C++ lessons learned from CMB." Furthermore, we will work on finding new but similar execution platforms to replace or supplement the XU3 (See Section 2), and on making the measurements of execution time and energy for a submitted program more precise.

References

[AMO93] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network flows: theory, algorithms, and applications*. Englewood Cliffs, N.J: Prentice Hall, 1993.

[Orl97] J. B. Orlin. “A polynomial time primal network simplex algorithm for minimum cost flows”. In: *Mathematical Programming* 78.2 (1997).

[AMO08] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. “Minimum Cost Flow Problem”. In: *Encyclopedia of Optimization*. Springer, Boston, MA, 2008.

[RML08] M. A. Revilla, S. Manzoor, and R. Liu. “Competitive Learning in Informatics: The UVa Online Judge Experience”. In: *Olympiads in Informatics* 2 (2008).

[Iha+10] P. Ihantola et al. “Review of Recent Systems for Automatic Assessment of Programming Assignments”. In: *Proc.of the 10th Koli Calling Int’l Conf. on Computing Education Research*. ACM, 2010, pp. 86–93.

[Ens+11] E. Enström et al. “Five years with Kattis: Using an automated assessment system in teaching”. In: *Frontiers in Education Conference (FIE)*. Oct. 2011.

[BDM12] R. E. Burkard, M. Dell’Amico, and S. Martello. *Assignment problems*. Revised reprint. Philadelphia: SIAM, 2012.

[CN13] J. M. Cebrian and L. Natvig. “Temperature effects on on-chip energy measurements”. In: *2013 International Green Computing Conference*. 2013.

[Int14] Intel. *Green Coding Contest*. 2014.

[FS15] T. Follan and S. Støa. *Climbing Mont Blanc: A Prototype System for Online Energy Efficiency Based Programming Competitions on ARM Platforms*. MSc Thesis, NTNU. 2015.

[Gre16] B. Gregg. “The Flame Graph”. In: *Commun. ACM* 59.6 (May 2016), pp. 48–57.

[Mag16] S. Magnussen. *Improving System Usability of Climbing Mont Blanc: An Online Judge for Energy Efficient Programming*. MSc Thesis, NTNU. 2016.

[Ing17] F. P. Ingebrigtsen. “Climbing Mont Blanc: Back-end Improvements”. MSc Thesis, NTNU. Trondheim: NTNU, 2017.

[EEC] EECS. *Energy Efficient Computing Systems*. `https://ntnu.edu/ie/eecs`.

[HiK] HiKey. *HiKey 960 Doc*. `https://www.96boards.org/product/hikey960/`.

[Nat] L. Natvig. *Climbing Mont Blanc*. `https://ntnu.edu/idi/lab/cal/cmb`.

[Nat+] L. Natvig et al. *Climbing Mont Blanc: A Training Site for Energy Efficient Programming on Heterogeneous Multicore Processors*. arXiv: `1511.02240 [cs.DC]`.

[Odr] Odroid. *Odroid-XU3 Wiki*. `http://odroid.com/dokuwiki/doku.php?id=en:odroid-xu3`.

[Pek] Peking University. *PKU Judge Online*. `http://poj.org/`.

[Sam] Samsung. *Exynos Wiki*. `https://en.wikipedia.org/wiki/Exynos`.