NOAA Technical Report NOS NGS 71

The Gravimeter Scheduling Problem: Proof of NP-Completeness and Approaches to Optimization

Dru Smith[1]
Amanda Bright [2]
Jerrad Martin [2]
Derek van Westrum[1]

Silver Spring, MD

November 23, 2019

1 National Geodetic Survey
2 National Geospatial-Intelligence Agency

Versions

Date	Changes
November 23, 2019	Original Release

The views expressed in this publication are those of the authors and do not imply endorsement by the Department of Defense or the National Geospatial-Intelligence Agency.

Approved for public release, 17-057

1 List of uncommon mathematical symbols

This paper uses some mathematical symbols which are not commonly found in geodetic texts. As such, a brief list of less common symbols and their descriptions are below.

Symbol	**Name**	**Description**
$\lceil x \rceil$	Ceiling *x*	The smallest integer equal to or greater than x
$\lfloor x \rfloor$	Floor *x*	The largest integer equal to or less than x
$n\%k$	*n* modulo *k*	The integer remainder after performing n/k on two integers n and k
$\binom{n}{k}$	*n* choose *k*	The number of combinations of k elements drawn from a set of n elements without order or repetition
$\lvert E \rvert$	Cardinality of E	The number of elements in a set E
$\forall$	For all	The situation to the left of ∀ is true for all values to the right of ∀

2 Prologue

Ad hoc comparisons of gravity instruments presumably go back to the earliest days of inclined planes and pendulums. The first, modern comparison of absolute freefall gravimeters, however, took place in Sevres, France in 1981 under the auspices of the International Bureau of Weights and Measures (BIPM). [Boulanger *et al*. 1983]. Since then, as the number of participating gravimeters grew, the philosophy of how to design an appropriate comparison schedule has continued to evolve: sometimes with a rigorous mathematical footing, and at other times with a less satisfying approach employing criteria that "feel right".

The purpose of this paper is twofold: First, to formally define the problem of optimizing an absolute gravimeter comparison schedule and describe its complexity. It will be clear then that a comprehensive solution is beyond the scope of this work. As a second goal though, we are hopeful that the results and techniques presented here begin to form a common framework and vocabulary for describing, designing, and assessing instrument comparisons. We strive to produce recommendations, but we also acknowledge that is necessarily an ongoing effort. We are excited to see members of the community take on the challenge of other aspects of this interesting problem.

3 Introduction

An absolute gravimeter (which will be called a ***meter*** for simplicity) is a portable scientific instrument capable of very precisely measuring the absolute acceleration of gravity (precision at the 0.0000001 m/s^2 level). Unfortunately, there is no standard against which these meters can be calibrated. In the absence of such calibration, a method has been devised to check gravimeters against one another in large groups to ensure they are in good working condition, at least relative to the group behavior. Groups of meters are therefore periodically brought to a laboratory and an "absolute gravimeter comparison" is held over a period of a few days.

At such a laboratory that hosts an "absolute gravimeter comparison" – called the ***event*** from this point forward - a number of gravity measuring piers will exist (being nothing more than special concrete slabs in the floor, built to be as insensitive as possible to building vibrations). These will be referred to as ***piers*** throughout the paper.

The mechanics of such an event are as follows: On any given day[1], some (possibly all) of the meters in attendance are set up, one meter per pier (if the number of meters is less than or equal to the number of piers, then each meter will be set up to run that day on some specified pier). The meters run all day and then stop. The operation of one meter on one pier for one day yields one measurement of gravity by that meter on that pier, and will be referred to as an ***observation***.

After the first day's observations are complete, the piers are vacated, the meters shuffled around to new piers, and the meters set up to run for a second day. Although there are many complicated methods for "comparing" two meters during an event, the standard definition is this: in order for two meters to be "compared" to one another they must have been set up on the same pier, but on different days. Such a comparison between two meters on one pier at two different days will be called one ***check***. The number of days separating the meters in one check is not relevant, as long as the check occurs within the duration of the event. For the purposes of this paper, it will be assumed that there is no such thing as a check "across piers", only "across days".[2] As such, each pier will generate some number of checks, depending on how many days the overall event lasts. Thus the total number of checks that the event generates will be the sum of all checks between each pair of meters.

[1] The word "day" here is used to mean a 24 hour period. In most such events, the setup occurs in the morning and the meters are left to run overnight into the next calendar day. To identify each "day" in an event, the calendar day of the *setup* will be used, even though the meters will run for the better part of "a day" and cross midnight into the next calendar day.

[2] This is not strictly true, as there are many ways to construct the least squares adjustment which processes a gravimeter comparison. NGS is planning a future study on this topic, but as of today, the standard procedure for all groups who regularly perform gravimeter comparisons is to consider only checks "on pier, across days".

This setting up of meters on piers continues for a number of days until some criteria are met and the event ends. Such criteria might be:

1) A pre-set number of days has passed, or
2) Every meter has at least one check against every other meter, or
3) Every meter has made at least some minimum number of observations, or
4) Some other more complicated criteria.

For the purposes of this paper, it will be assumed that criteria 3 is the definitive "end of event" trigger. This is part of a "recommended practice" as proposed in this paper, but is in conflict with current practice. A complete examination of both practices, including a justification of recommended practice is found in Appendix C.

Finally, it is important to note that such events are expensive in both time and money, almost always involving international travel, the transportation of expensive equipment and continuous long days of work without any days off. As such, the problem of *optimizing* the schedule has more than an academic interest, but is of genuine financial consideration. With that in mind, consider that a meter can be checked against itself at any time outside of an event, and that allowing a so-called "self-self" check to occur during an event means sacrificing an otherwise possible (and significantly more important) check between two different meters. Therefore, one of the two requirements of optimization will be to eliminate (or at worst, minimize) self-self checks in the schedule. The second goal will be to have the most diverse distribution (defined later) of checks among all meters at the event. The creation of an optimized schedule will be called the gravimeter scheduling problem or **GSP**.

The GSP has similarities to others, such as the Oberwolfach problem (Lenz and Ringel, 1991), the Nurse Scheduling Problem (Solos et al, 2013), or the TimeTable problem (Gunawan et al, 2007) none of which have a generalized solution, and thus the paper will discuss the success of computational algorithms that seek optimized solutions.

This paper begins by providing definitions and equations in Section 4, including what it means for a solution to be "optimized", as well as a special (only occasionally achievable) case called the "perfect" solution. It will then prove the NP-completeness property of the problem in Section 5. In Section 6, the feasibility of creating a perfect schedule for an NP-complete problem will be discussed. Section 7 examines the magnitude of the search set, and Section 8 discusses three approaches to finding the optimal solution. The paper closes with conclusions in Section 9 and an outline of future work in Section 10.

4 Definitions

4.1 Defining the event

There are many variables which can affect how the event plays out. Because modifying any of these variables effectively defines a new mathematical problem, the following restrictions and/or assumptions will be adhered to throughout this paper.

The first restriction is that any schedule must be physically possible, defined by two rules:

1) On a single day, no single meter may be scheduled on more than one pier.

2) On a single day, no single pier can hold more than one meter.

Let m be the number of meters which show up for the event. Let p_A be the number of available piers, and p be the number of piers actually used. If p_A is larger than m, the event will be restricted only to the first m piers. Thus:

$$p = \min(m, p_A). \quad (1)$$

With this rule in place, it is possible to say definitively that, for every event being considered in this paper, $m \geq p$. It is assumed that all meters are in good working condition (and thus available to be scheduled for an observation on any day of the event). Also, all piers will be in good working condition (and thus available for a meter to be scheduled upon them for an observation on any day of the event).

There are other restrictions to the problem.

- The event ends after all meters have made at least o observations.
- No meter shall make more than (o + 1) observations during the event.
- No pier shall be vacant during any days of the event.
- The event should take the fewest days possible, and not skip any days.

Specifically, the number of days, d, in the schedule will have be:

$$d = \left\lceil \frac{mo}{p} \right\rceil. \quad (2)$$

These events have been occurring since the early 1990s, and have grown in size over the decades. Meters frequently outnumber piers, with a typical event being 10-20 meters over 5-15 piers with 3-4 observations each (e.g. Newel et al. 2017; Pálinkáš et al. 2017). The largest thus far was the International Comparison of Absolute Gravimeters 2017 (ICAG17) which took place in China with 28 meters, 9 piers, and 4 observations per meter (results not yet published[3]).

This section closes with a summary list which defines an event for this paper. It should be noted that modifying any of these will define a new event and a thus pose a new Gravimeter Scheduling Problem.

Definition of an event consisting of m meters with p piers:

1) All daily assignments of some meter on some pier must be physically possible (see above).
2) All meters must make at least o observations.
3) The duration of the event, d, shall be the minimum needed for all meters to make o observations.
4) None of the piers can be vacant on any day.
5) Any meter can be scheduled on any pier on any day.

Certain additional rules will naturally fall out of the above definition. For example, numbers 2, 3 and 4 will always mean:

a) No meter will make more than o+1 observations.

[3] As work progresses on this report, it should be posted at this website: https://kcdb.bipm.org/AppendixB/KCDB_ApB_info.asp?cmp_idy=1615&cmp_cod=CCM.G%2DK2.2017&page=

b) At least 1 and at most *m* meters will make *o* observations.
c) At least 0 and at most *m*-1 meters will make *o*+1 observations.

With this definition in hand, and given an event with some given *m*, *p* and *o* combination many (many!) schedules could be drawn up. The problem is to both define what it means for a schedule to be optimized and then determine how best to find that optimized schedule. This will be presented in the next section.

4.2 Defining the Problem

As mentioned earlier, gravimeter comparisons are expensive. Thus, the problem is one of getting the "best" results out of the event. Therefore the problem to be solved is this: given all possible schedules for an event, define the optimized schedule and find it quickly. In order to do this, it is necessary to introduce some definitions.

Definition 1: The ***schedule*** is a *p* × *d* matrix, where each element of the matrix will be the meter number {1, 2, …, *m*} of the meter occupying that pier (*p*, row) on that day (*d*, column).

An example of a schedule with (*m, p, o*) = (6, 3, 2), which gives *d* = 4 days, is shown in the orange squares of Figure 1.

		Days			
		1	2	3	4
Piers	1	1	4	2	6
	2	2	5	3	4
	3	3	6	1	5

Figure 1: A schedule with 6 meters, 3 piers and a required minimum of 2 observations per meter.

Definition 2: The ***check matrix*** is the upper triangular portion of an *m* × *m* matrix showing checks that occur between meters based on a given schedule. Every element will be an integer, reflecting the number of checks between meters. Let *n* be the number of non-diagonal elements of the check matrix. Note that the *sum* of all elements (*n*+*m*) of a check matrix is *C*, the sum of all non-diagonal elements (*n*) of a check matrix is *C** and the sum of all diagonal elements (*m*) is *S*.

An example of a check matrix, which reflects the schedule in Figure 1, is provided in Figure 2.

		Second Meter					
		1	**2**	**3**	**4**	**5**	**6**
First Meter	**1**	0	1	1	1	1	2
	2		0	1	2	1	1
	3			0	1	2	1
	4				0	1	1
	5					0	1
	6						0

Figure 2: The check matrix generated by the schedule in Figure 1. In blue are the *n* (=15) non-diagonal elements, which sum up to *C (=18). In red are the *m* diagonal elements which sum up to *S* (=0). The sum of all elements is *C* = *C** + *S* (=18)**

As mentioned earlier, it is often considered a waste of valuable resources to set up a check between a meter and itself during such an event. As such, no meter should complete a check against itself, unless it is unavoidable given the parameters (and it *is* sometimes unavoidable). Let *S* be the number of self-self checks during any chosen schedule for the event. Also, let S_0 be the *minimum number of unavoidable of self-self checks* during any given event. (The equation for S_0 is given later). This means that no schedule can be designed with less self-self checks than S_0, or:

$$S_0 \leq S \; \forall \; S. \tag{3}$$

Because no piers are ever vacant, and all possible schedules for a given event last the same number of days, the total number of checks in an event is constant, no matter what schedule is chosen. Let *C* be that total number of checks in the event, computed as follows: The number of checks on any single pier is $\binom{d}{2}$. As checks only occur on a pier (and not across piers) and because no pier is ever empty, the total number of checks in the event is the number of checks on one pier times the number of piers, specifically:

$$C = p\binom{d}{2} = \frac{pd(d-1)}{2}. \tag{4}$$

Next let *C** be the total number of *non self-self checks* for any given schedule. That is:

$$C^* = C - S. \tag{5}$$

Unlike *C*, the value *C* does* depend on the schedule chosen, because it depends upon how many self-self checks (*S*) exist in the chosen schedule. A related value is C_0^*, the total number of non-self-self checks in the event *if* the number of self-self checks has been minimized:

$$C_0^* = C - S_0. \tag{6}$$

In order to define the optimization of the schedule a few other items are still needed. First, it is necessary to know the number of *possible* pairings between all meters. Let *n* be the total number of ways to pair any two of the *m* meters, without pairing a meter with itself. These pairings may be called ***potential checks*** to reflect the fact that, if the schedule is set up long enough and properly, a check could be made covering every one of these pairs of meters. The equation for *n* is simply $\binom{m}{2}$:

$$n = \binom{m}{2} = \frac{m(m-1)}{2}. \tag{7}$$

Some final definitions must be introduced to complete the discussion.

Definition 3: The ***score***, designated σ, of a schedule is the standard deviation of all non-diagonal elements of the check matrix.

At this point, it is helpful to introduce three different sets of schedules. Let *E* be the set of all possible schedules for any given (*m,p,d*) combination[4]. Let E_1 be the subset of *E* which only contains schedules where the number of observations made by each meter is either *o* or *o*+1. That is, E_1 contains all possible schedules for any given (*m,p,o*) combination. Let E_2 be the subset of E_1 which only contains schedules where $S=S_0$.

Definition 4: The ***optimized score***, designated σ_0, of any given (*m,p,o*) combination is the smallest possible score among all schedules in the set E_2 (e.g. $S=S_0$).

With these definitions in hand, the definition of an optimized schedule may finally be given.

Definition 5: An ***optimized schedule*** is one where the check matrix fulfills the following criteria:

1) The sum of diagonal elements is minimized: $S = S_0$,and
2) The score is the optimized score: $\sigma = \sigma_0$.

The set of all optimized schedules will be called E_3, and is that subset of E_2 which contains all schedules with the optimized score. Any element of set E_3 is a solution to the GSP. See the diagram below.

[4] The (*m,p,d*) combination is introduced first, rather than (*m,p,o*), purposefully. Given some (m,p,o) combination, the number of days, *d*, is fixed immediately, through equation 2. Once *d* is known, most computer algorithms searching for an optimized schedule (defined later) will tend to work with *m*, *p* and *d* as their primary variables, which puts them in set *E*. Without a bound of *o* or *o*+1 on observations, this means set *E* can be substantially larger than set E_1. As such, a good search algorithm, while initially bounded by set *E*, should use the *o* or *o*+1 bound to keep itself within set E_1, and also S0 to keep itself within the set E_2.

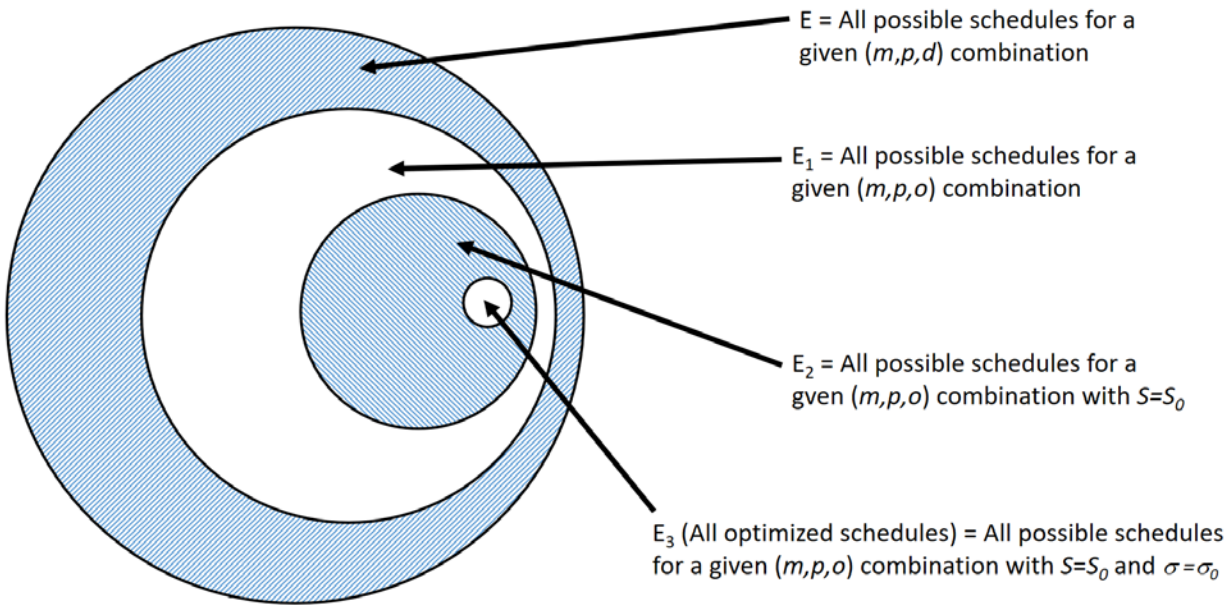

Figure 3: The sets *E*, E_1 , E_2 and E_3 of various schedules

<u>*At least one optimized schedule always exists*</u>. That is, of all possible (*m,p,o*) schedules (E_1), there always exists a subset (E_2) that have the property $S=S_0$, and in that subset there must be at least one schedule which has the smallest score. That smallest score is, by definition, the optimized score, and so E_3 must exist with at least 1 entry.

The ***Gravimeter Scheduling Problem (GSP)*** is therefore defined as the problem of finding an optimized schedule for an event as defined in Section 3.1, given *m*, *p* and *o*.

Quickly finding an optimized schedule was the driving force behind this study. Thankfully, there are certain known quantities, which bound the problem before searching. The first of these is S_0 which can be pre-computed (see next section), and thus the search for the optimized schedule can be limited to the (generally much smaller) set E_2 and not the greater set E_1. The second bound on the search for an optimized schedule has to do with something called the "perfect score", introduced next.

<u>Definition 6</u>: A ***perfect score***, designated σ_P, is the smallest standard deviation computable when a set of *n* integers must sum up to C_0^*.

In the above example used in Definition 1 and Definition 2, (*m,p,o*) = (6,3,2) yields *d*=4 and (*n,C*) = (15,18). [Also in that example, the schedule shown has *S*=0 and *C**=18. Although the formula for S_0 is net yet given, Figures 1 and 2 show a schedule where there are no self-self checks . Thus it is proven that, for this *m,p,o* combination, $S_0 = 0$ and therefore C_0^* is equal to 18.] So, the perfect score in this example comes from identifying which combination of 15 (*n*) integers which sum up to 18 (C_0*) will yield the smallest standard deviation. The answer turns out to be "twelve 1's and three 2's", which leaves a standard deviation of 0.414. No other combination of 15 integers summing up to 18 has a smaller standard deviation than 0.414. So in this example a perfect score would be $\sigma_P = 0.414$.

It is critical to note the following: The perfect score is always *computable*, being just a function of two known integers (*n* and C_0*), which are in turn just functions of *m*, *p* and *o*. But it is *not* always possible to design a schedule whose score is equal to the perfect score.

Put another way: For any given (*m,p.o*) combination, there will *always* be at least one optimized schedule whose score is the optimized score, but that optimized score is not always as small as the perfect score. Thus:

$$\sigma_P \leq \sigma_0 \tag{8}$$

The perfect score is introduced as a second (imperfect) bound on the search for an optimized schedule, and it works like this: Because the *optimized* score is *not* pre-computable, any search algorithm looking for it cannot know if it has found it, and thus does not know to stop searching. But, since the *perfect* score *is* pre-computable, and given equation 8, any search algorithm which finds a schedule with the *perfect* score *does* know that it has *also* found the *optimized* score, and knows to stop searching.

An interesting, and important quality of the perfect score is presented in the next two theorems.

Theorem 1: If a set of *n* integers sum to C_0^* and that set has a score that is equal to the perfect score ($\sigma = \sigma_P$), and C_0^* is evenly divisible by *n*, then the perfect score is exactly zero ($\sigma_P = 0$) and all elements of that set are equal to one integer I_{LO}, which fulfills the following equation:

$$I_{LO} = \frac{C_0^*}{n} \tag{9}$$

Proof: The proof is trivial. First, it is self-evident that summing *n* values of I_{LO} from equation 9 will yield C_0^*. It is further self-evident that any set of identical numbers will have a zero standard deviation. As zero is the smallest possible standard deviation, it is also the perfect score for this (*n*, C_0^*) combination.

Theorem 2: If a set of *n* integers sum to C_0^* and that set has a score that is equal to the perfect score ($\sigma = \sigma_P$), and C_0^* is *not* evenly divisible by *n*, then that set will consist exactly of N_{LO} instances of integer I_{LO} and N_{HI} instances of I_{HI}, which fulfill the following equations:

$$I_{LO} = \left\lfloor \frac{C_0^*}{n} \right\rfloor \tag{10}$$

$$I_{HI} = I_{LO} + 1 \tag{11}$$

$$N_{LO} = n - (C_0^* \,\%\, n) \tag{12}$$

$$N_{HI} = n - N_{LO.} \tag{13}$$

Proof: Consider all possible sets of *n* integers which sum to C_0^*. Let each integer element of the set be called I_i. The mean and standard deviations of these integers is:

$$\bar{I} = \left(\sum_{i=1}^{n} I_i\right)/n = \frac{C_0^*}{n} \tag{14}$$

$$\sigma = \sqrt{\sum_{i=1}^{n} \frac{(I_i - \bar{I})^2}{n-1}}. \tag{15}$$

Because every I_i value is an integer, there are only a limited number of values which can occur in the numerator of equation 15, and of all possible values, the two smallest possible values come the two integers nearest to $\bar{I}$, which are:

$$I_{LO} = \left\lfloor \frac{C_0^*}{n} \right\rfloor \tag{16}$$

$$I_{HI} = \left\lceil \frac{C_0^*}{n} \right\rceil = I_{LO} + 1. \tag{17}$$

It is self-evident that the smallest value computable from equation 15 (that is, the perfect score), can be computed if the numerator were restricted to only its smallest values. Therefore, if one restricts I_i solely to the two integers nearest $\bar{I}$, *and* is further restricted that all values of I_i must sum to C_0^*, then one and only one possible set of *n* integers is possible, being the set described in equations 10-13.

Corollary 1: The perfect score of a set of *n* integers which sum to C_0^*, when C_0^* is not evenly divisible by *n* is:

$$\sigma_P = \sqrt{\frac{(n - N_{LO})N_{LO}}{(n-1)n}}. \tag{18}$$

Proof: The proof is a fairly straightforward algebraic exercise beginning with equation 15 and relying upon equations 10-14 and is therefore not presented herein.

With simple methods of computing the perfect score, the following definition now is introduced.

Definition 7: A **perfect schedule** is one where the check matrix fulfills the following criteria:

1) The sum of diagonal elements is minimized: $S = S_0$, and
2) The score is the perfect score: $\sigma = \sigma_P$.

As mentioned earlier, an *optimized* schedule is always achievable, while a *perfect* schedule is only *occasionally* achievable. A discussion about the achievability of perfect scores is provided later in the paper.

In order for a schedule to be optimized, it needs minimal (preferably eliminated) self-self checks. To calculate the minimum number of unavoidable self-self checks for any given *m,p,o* combination, the third theorem will be defined.

Theorem 3: Any schedule, with given values of *m*,*p* and *o* will require no fewer than

$$S_0 = m\left[\left(p-(o\%p)\right)\frac{1}{2}\left\lfloor\frac{o}{p}\right\rfloor\left(\left\lfloor\frac{o}{p}\right\rfloor-1\right)+(o\%p)\frac{1}{2}\left\lfloor\frac{o}{p}\right\rfloor\left(\left\lfloor\frac{o}{p}\right\rfloor+1\right)\right]+(pd-mo)\left\lfloor\frac{o}{p}\right\rfloor \quad (19)$$

self-self checks.

The inside of the first term counts the self-self comparisons for one meter in the schedule which is then (by multiplying by *m*) used to find the number of self-self checks for all meters. The last term accounts for the self-self comparisons that arise from the extra observations that some meters may require in the schedule.

Proof: Given (*m, p, o*), begin by only considering that part of the schedule where each meter makes exactly *o* observations. Under such a schedule, it always happens that a meter will be scheduled $\left\lfloor\frac{o}{p}\right\rfloor$ times on *(p-o%p)* piers and will be scheduled $\left\lfloor\frac{o}{p}\right\rfloor+1$ times on *o%p* piers. Further, when any meter appears on the same pier *k* times, there will $\binom{k}{2}$ self-self checks of that meter on that pier. Thus does each meter, after having made *o* observations, generate $\frac{1}{2}\left\lfloor\frac{o}{p}\right\rfloor\left(\left\lfloor\frac{o}{p}\right\rfloor-1\right)$ self-self checks on $\left(p-(o\%p)\right)$ piers, and $\frac{1}{2}\left\lfloor\frac{o}{p}\right\rfloor\left(\left\lfloor\frac{o}{p}\right\rfloor+1\right)$ self-self checks on $(o\%p)$piers. As such, before considering meters making *o+1* observations, each meter will have

$$\left(p-(o\%p)\right)\frac{1}{2}\left\lfloor\frac{o}{p}\right\rfloor\left(\left\lfloor\frac{o}{p}\right\rfloor-1\right)+(o\%p)\frac{1}{2}\left\lfloor\frac{o}{p}\right\rfloor\left(\left\lfloor\frac{o}{p}\right\rfloor+1\right) \quad (20)$$

self-self checks. Multiplying equation 14 by *m* gives a count of self-self checks before considering meters making *o+1* observations. This yields the first half of equation 19.

However, in the case that $pd - mo \neq 0$, some meters will have $o + 1$ observations which can result in additional self-self checks. To count those, *pd-mo* meters will have an additional observation. Each meter's one additional observation must be scheduled on a pier that has $\left\lfloor\frac{o}{p}\right\rfloor$ observations by that meter already. Thus a meter's additional observation on that pier will obviously generate $\left\lfloor\frac{o}{p}\right\rfloor$ additional self-self checks of that meter. As this happens to *pd-mo* meters, the number of additional self-self checks generated by the meters making o+1 observations is:

$$(pd-mo)\left\lfloor\frac{o}{p}\right\rfloor \quad (21)$$

This yields the second half of equation 19.

The following example illustrates how some schedules cannot avoid self-self checks.

Example 1: An example of an event that cannot avoid self-self checks, even when optimized, is (*m, p, o*) = (3, 3, 4). This event requires *d* = 4 days (equation 2) and S_0 = 3 self-self checks (from equation 19) at a minimum, with $C_0^* = C - 3 = 18 - 3 = 15$. Accepting that there must be 3 non-zero diagonal elements, the perfect score, based on the need to sum 3 (*n*) integers to 15 (C_0^*), is 0 (because it can be done with three 5's, yielding a standard deviation of 0; it so happens that in this example, a perfect schedule is achievable.) One sample (of many possible) of a perfect (and thus optimized) schedule within this

example, and its check matrix follows in Figure 4 and Figure 5. Any search algorithm which found this (or another) perfect schedule would know it had reached the optimized schedule, and could stop.

		Days			
		1	2	3	4
Piers	A	1	2	3	1
	B	2	3	1	2
	C	3	1	2	3

Figure 4: An example of a (*m,p,o*)=(3,3,4) perfect (and, by extension, optimized) schedule.

		Second Meter		
		1	**2**	**3**
First Meter	**1**	1	5	5
	2		1	5
	3			1

Figure 5: The check matrix generated by the schedule in Figure 4. In blue are the *n* (=3) non-diagonal elements, which sum up to *C (=15). In red are the *m* (=3) diagonal elements which sum up to *S* (=3). The sum of all elements is *C* = *C** + *S* (=18). Note that the diagonal (red) elements, while not zero, are minimized (*S* = S_0 = 3). The score of the blue elements is perfect ($\sigma = \sigma_0 = 0$) and thus this is the check matrix of a perfect (and thus optimized) schedule, without zeroes on the diagonal. Note that a perfect score isn't always zero, but just happens to be so in this case.**

5 NP-Completeness

Because of its relation to other optimization problems, such as the Timetable (TT) problem, it was suspected that the GSP was in a class of problems called "Non-Parametric Complete" or NP-Complete. Such problems have well-developed but not perfect computation strategies to search for optimizing solutions, so it was considered worthwhile to first determine if the GSP were NP-Complete.

Even et al (1976) proved that a timetable problem (TT) is NP-complete. It will be shown that the TT can be reduced to GSP, and thus the GSP is NP-complete. Proving that this problem is NP-complete proves any computer algorithm will require exponential time to solve the problem. For an introduction to NP-completeness, Manber (1989) is an appropriate reference.

The known NP-complete timetable problem finds schedules for teachers to cover classes, given the various class schedules and each teacher's particular availability as defined by Even et al. (1976) is given next.

Theorem 3 (Timetable problem, TTP): Given a finite set *H* (of hours in a week), a collection $\{T_1, T_2, ..., T_n\}$ in which $T_i \subseteq H$ (there are *n* teachers and T_i is the set of hours during which the *i*th teacher is available for teaching), a collection $\{C_1, C_2, ..., C_m\}$ in which $C_j \subseteq H$ (there are *m* classes and C_j is the set of hours during which the *j*th class is available for studying), and an $n \times m$ matrix *R* of nonnegative integers (R_{ij} is the number of hours which the *i*th teacher is required to teach the *j*th class), the TT is the problem of determining whether there exists a function

$$f(i, j, h) : \{1, ..., n\} \times \{1, ..., m\} \times H \rightarrow \{0, 1\} \tag{22}$$

such that

a) $f(i,j,h) = 1 \Rightarrow h \in T_i \cap C_j$;
b) $\sum_{h \in H} f(i,j,h) = R_{ij} \quad \forall\, i \in [1,n] \text{ and } j \in [1,m]$;
c) $\sum_{i=1}^{n} f(i,j,h) \leq 1 \;\; \forall\, j \in [1,m] \text{ and } h \in H$;
d) $\sum_{j=1}^{n} f(i,j,h) \leq 1 \;\; \forall\, i \in [1,n] \text{ and } h \in H$;

Specifically, (a) assures that a meeting takes place only when both the teacher and the class are available, (b) assures that the number of meetings during the week between teacher *i* and class *j* is the required number R_{ij}, (c) assures that no class has more than one teacher at a time, and (d) assures that no teacher is teaching two classes simultaneously.

With this in mind, we define the Gravimeter Scheduling Problem (GSP) and prove it can be mapped to the TTP.

Definition 8 (The Gravimeter Scheduling Problem, GSP): Given a finite set *M* (meters), a finite set *P* (piers), a finite set *D* (days), a class of $|M| \times |P|$ matrices of $\mathcal{R}$ of nonnegative integers (R_{mp} represents the number of times the *m*th meter can be assigned to the *p*th pier). To further describe *R* in $\mathcal{R}$, it is a matrix that must meet the following constraints:

1. Each entry is $\left\lfloor \frac{o}{|P|} \right\rfloor$ or $\left\lfloor \frac{o}{|P|} \right\rfloor + 1$, where o=$\left\lfloor \frac{|P| \cdot |D|}{|M|} \right\rfloor$;
2. Each column sums to $|D|$; and
3. ($|P|$-$|P| \cdot |D|$+$|M| \cdot o$) rows sum to *o* and ($|P| \cdot |D|$-$|M| \cdot o$) rows sum to (*o*+1).

Lastly, given a collection H_{md} in which $H_{md} \subseteq P$ (H_{md} is the set of piers to which the *m*th meter can be assigned on the *d*th day). The scheduling problem is determining whether there exists a function

$$f(m, p, d) : M \times P \times D \rightarrow \{0, 1\} \tag{23}$$

such that

a) $f(m,p,d) = 1 \Rightarrow p \in H_{md}$;
b) $\sum_{d \in D} f(m,p,d) = R_{mp} \quad \forall m,p \text{ where } R \in \mathcal{R}$
c) $\sum_{m \in M} f(m,p,d) \leq 1 \quad \forall p,d$
d) $\sum_{p \in P} f(m,p,d) \leq 1 \quad \forall m,d$

The function *f* is defined such that *f* (*m, p, d*) = 1 if the *m*th meter is placed at the *p*th pier on the *d*th day and *f* (*m, p, d*) = 0 otherwise. The restrictions of *f* are some of the constraints of the problem. More specifically, (a) assures that gravimeters are only placed in available piers, (b) ensures that self-self

comparisons either do not exist or are minimal by controlling the number of times a gravimeter can appear on any pier (i.e. *R* determines how many observations a meter *m* makes at a pier *p*) (c) ensures there is no more than one observation of a gravimeter on a pier on any day and (d) ensures that a meter is assigned to no more than one pier a day.

Evan et al (1976) showed that the TTP is NP-complete. The following will prove that the GSP is also NP-complete because the problems can be directly related.

Proof: Given a schedule from the TTP, it is possible to check whether or not the corresponding function (equation 22) meets the parameters defined in Definition 8. To transform the TTP into the GSP, correspond the set {1*, ..., n*} of teachers to the set *M* of gravimeters, the set {1*, ..., m*} of classes to *P* of piers, and the set *H* of hours to *D* of days. Then $H_{md} = \{ j \mid h \in T_i \cap C_j \}$. It is possible to correspond (a), (b), (c), and (d) in the TT to (a), (b) where one $R \in \mathcal{R}$ is chosen, (c), and (d) in GSP, respectively. Thus the TTP corresponds to the GSP and therefore the GSP is NP-Complete.

A few words about the above proof may add clarity: The proof represents a simplified version of the GSP. The proof defines the GSP as the creation of physically possible schedules with minimal self-self checks. It concludes that finding those (physically possible with minimal self-self checks) schedules is NP-Complete based on the TTP. Outside of the proof, we are additionally searching for the "best" schedule(s) from those derived from an NP-Complete problem based on maximizing piers for a meter. But that additional optimization across an NP-Complete problem doesn't change the fact that the base GSP is NP-Complete. That is: the TTP has been proven to be NP-Complete. We are able to create a mapping from the (simplified) GSP to the TTP. Our proof shows that the GSP is at least as hard as an NP-Complete problem, therefore it is NP-Complete. In later sections additional optimization constraints, such as minimizing travel costs, are discussed, though not solved in this paper. Such additional optimization constraints are not expected to change the NP-completeness of this problem, but no proof of that hypothesis has been sought as yet.

This proof illustrates that the GSP is NP-complete. And while that is an interesting conclusion, the more pressing issue is to find an optimized solution to the GSP. By the nature of NP-completeness, this means it takes exponential time to find such a schedule.

6 Feasibility of a Perfect Schedule

Because the optimized score cannot be pre-computed, but the perfect score can, it is worth discussing whether, for a given event, it can be determined whether the perfect schedule exists at all. Such knowledge can assist in bounding the algorithms searching for the optimized schedule.

To begin, it will be shown that there are cases in which it is clear the perfect schedule is *not* possible.

Theorem 4: If *pd-mo* ≥ 2 and 2(o+1)-p ≥ $\left\lceil \frac{C}{n} \right\rceil + 1$, the perfect score is unattainable.

Proof: The value *pd-mo* gives how many piers will be vacant on the last day of observations after all *m* meters have been assigned to collect their *o* observations. Since this paper has assumed that no piers will be vacant on any day, this means there will be *pd-mo* meters which will have to take an extra

observation on that last day. Therefore when *pd-mo* ≥ 2 it means that at least two meters will have *o*+1 observations. Now, consider the two meters that will make *o*+1 observations.

They must both be placed on 2(*o* + 1) − *p* piers, giving them the same number of checks. However,

$$2(o+1) - p \geq \left\lceil \frac{C}{n} \right\rceil + 1 \tag{24}$$

which means 2(o+1)-p is a third value in the check matrix. All other check values on the check matrix will be (2*o*+1-*p*) between meters of *o* and (*o*+1) observations and (2*o*-*p*) between meters each with o observations. In order to obtain a perfect score, only one or two values will appear on the check matrix, based on parity of the parameters. By proving a third value must exist, it is apparent that the perfect score is unattainable.

Unfortunately, while this predictor is true, it is not a perfect predictor of every instance when the perfect score is unattainable. See Appendix A. The case of (*m,p,o*)=(5,3,2) does fulfill the inequalities of Theorem 4, and brute force has proven that indeed this combination cannot achieve the perfect score. However the case of (*m,p,o*)=(3,2,3) does *not* fulfill the two inequalities of Theorem 4, and yet its optimal score is not the perfect score.

Example 2: An example of a case that demonstrates Theorem 4 is (*m, p, o*) = (5, 3, 2). Equations 2, 4 and 7 imply the event will take *d*=4 days, have *C*=18 total checks and there are *n*=15 potential non-self-self checks. Equation 19 says that S_0=0, so there are no self-self checks required, and thus equation 6 says some schedule with C_0*=18 non-self-self checks is possible. Equations 10-13 state that 15 (*n*) values summing up to 18 (C_0^*) with the smallest standard deviation will consist of 12 (N_{LO}) 1's (I_{LO}) and 3 (N_{HI}) 2's (I_{HI}), for a perfect score of σ_P=0.414. That is, for a perfect score, the check matrix will contain 1's and 2's only, but *no* 3's. Yet, because there are 5 meters and only 3 piers, and because meters 1 and 3 both have 3 observations there must be 3 checks between these two meters. That is, there must be at least one "3" in the check matrix, for this *m,p,o* combination, and so for this example the perfect score is unachievable. One possible schedule and its check matrix are shown in Figure 6 and Figure 7.

		Days			
		1	2	3	4
Piers	A	1	4	3	5
	B	2	1	4	3
	C	3	2	5	1

Figure 6: An example of a (m,p,o)=(5,3,2) schedule.

		Second Meter				
		1	2	3	4	5
First Meter	1	0	2	3	2	2
	2		0	2	1	1
	3			0	2	2
	4				0	1
	5					0

Figure 7: The check matrix generated by the schedule in Figure 6. Note the "3" at row 1, column 3 (meaning meter 1 has 3 checks against meter 3)

A perfect schedule often cannot be attained for a variety of reasons. This means the search for an optimized schedule is often reduced to "best schedule found, within the computational time available" if a perfect solution cannot easily be found. Without a brute-force check of every schedule, these "best found" schedules can never be definitely stated as being "optimized schedules."

It is likely that other cases exist which either guarantee the achievability of a perfect score or guarantee the non-achievability of a perfect score, but further investigation down this line has been postponed for future studies.

7 Quantifying the number of schedules

Before investigating computational searches for the solution to the GSP, it will be instructive to consider how large the various search sets are. However, a word of warning is worthwhile here: the numbers of possible schedules quickly grows out of bounds for any sort of brute-force search algorithm. Therefore, while some startlingly large numbers are derived, and discussion of brute force is provided, the majority of this section is devoted to algorithms which seek an optimized schedule without tediously examining every *possible* schedule.

Section 3.2 defined the sets E, E_1, E_2 and E_3. All possible schedules fall inside E. Since o is also given, it is always possible to know if a schedule falls inside E_1. Further, since the value S_0 is pre-computable, it is also always possible to know if a schedule falls inside E_2. However, as σ_0 is (currently) not pre-computable it is (generally) impossible to know that a schedule falls in E_3. And since the solution to the GSP is found in E_3, only three circumstances exist which guarantee that the algorithm has found an optimized schedule:

1) Some method of pre-computing the optimized score is developed, and the search algorithm has found a schedule with the optimized score
2) The search algorithm has found a schedule with the perfect score
3) Every schedule in E_2 has been searched, and one with the smallest score is chosen.

Condition #1 is currently unsolved. Condition #2 will only occur sometimes, and no method yet exists which provides proof, for any given (*m,p,o*) combination whether the perfect schedule even exists. Condition #3 guarantees success, but quickly grows out of bounds. Therefore, as mentioned earlier, most search algorithms will default to "best solution found within an allowable search time".

However, before giving up on finding the optimized score, an investigation of Condition #3 is conducted.

7.1 Total number of schedules

From the standpoint of what is physically possible, it is an easy exercise to consider the cardinality of (total number of schedules in) set *E*. Begin with any (*m,p,o*) combination, and use equation 2 to compute the number of days, *d*.

Now consider how many ways the schedule can be arranged, just within a single day. On any given day, on pier 1, there are *m* possible meters which can occupy it. This leaves *m*-1 meters possible on pier 2, etc. Therefore, the number of possible schedules per day (SPD) is:

$$SPD = \frac{m!}{(m-p)!} \tag{25}$$

This same exercise may be repeated every day. If there is no regard for self-self checks or other restrictions (such as each meter performing *o* or *o*+1 observations) then each day's schedule is independent of the others. Therefore the number of schedules contained in set *E* is:

$$|E| = (SPD)^d = \left[\frac{m!}{(m-p)!}\right]^d = \left[\frac{m!}{(m-p)!}\right]^{\left\lceil\frac{mo}{p}\right\rceil} \tag{26}$$

This accounts for all possible *p*-permutations of *m* which may be repeated each day. It is clear that a brute force analysis of every schedule will not be feasible for even modest choices of (*m, p, o*) (e.g. $|E| = 2.6 \times 10^{64}$ for the (15, 11, 4) case). Note that while many of these schedules will produce identical check matrices, it is difficult to predict which to exclude from the analysis. This is related to the NP-completeness of the problem. In any case, such exclusions put very few additional (*m, p, o*) sets within reach.

The size of E_1 appears to be more difficult to compute and will not be investigated within this report, except when it can be directly counted, as in the brute force method (see later).

7.2 Strategies and rules

During the search for the solution to the GSP it is one thing to check and reject a schedule, but it is another thing altogether to know a-priori not to even bother checking a schedule at all. That sort of knowledge can be of substantial time savings. As such, a few general observations, strategies and rules will be mentioned below.

7.2.1 Minimize the Search Space

Although the solution to the GSP is a schedule, the actual search will be upon the parameters of the check matrix. And many different schedules can yield the exact same check matrix. If it can be pre-determined that a schedule will yield a check matrix which has already been seen, then the search algorithm should avoid even checking such a schedule.

In order to help instruct the algorithm, note two things:

1) Swapping two days in a schedule will not change the check matrix
2) Swapping two piers in a schedule will not change the check matrix
3) Swapping two meters *will* change the *positions of values* of the check matrix, but will not change its *S* and σ values.

If there were an efficient bookkeeping method to not only track schedules, but track combinations of pier-wise components and day-wise components of schedules, it might be possible to avoid going over effectively the same ground. For instance, what if an algorithm didn't search for "meters on piers" as a one-by-one check, but rather pre-computed all possible day combinations and/or pre-computed all possible pier combinations, and looked for schedules made up by investigating combinations of these whole piers at a time or whole days at a time? Such an approach warrants further investigation, but is not presented herein.

The only concrete rule which will come from the above observations, and was implemented in all of the search algorithms below is this:

Rule #1: The first day doesn't matter

Without loss of generality, day 1 can be set to any set of *p* meters. Although this seems a somewhat weak rule, consider the above case of (*m,p,o*) = (15,11,4). If the schedule chosen for day #1 can be any possible schedule, then the search reduces from 2.6×10^{64} schedules down to 4.8×10^{53}, a search set 54.5 billion times smaller than the original. While still too large to be easily managed, a reduction factor of 54.5 billion is worth using.

8 Search Algorithms

Three different algorithms were investigated for their ability to *search for* a solution to the GSP. Only 1 method (brute force) is guaranteed to find a solution. The other methods, as mentioned before, will be restricted to "finding an acceptable solution within reasonable time constraints." The inherent vagueness of the terms "acceptable" and "reasonable" is fully acknowledged, yet cannot be avoided when dealing with algorithms which cannot guarantee finding the solution to the GSP.

8.1 Optimized Schedule Search Algorithm #1: Brute Force

Brute force is a method applicable only to "very small" values of (*m,p,o*). It is the method of tediously building every possible schedule in *E*, checking to see if it is in E_1, and if so, seeing if it is in E_2 and if so, computing σ and keeping track of the smallest σ value found through this process. By checking all possible schedules, the algorithm guarantees success in finding σ_0, and thus the optimized schedule.

A subroutine "bruteforce.f"[5] was developed to apply this method, and run on all possible (within reasonable computational time limits) combinations of $2 \le m \le 6$, $2 \le p \le 6$, and $1 \le o \le 6$. Although designed with efficiency in mind, it "only" was capable of checking just under 4 million schedules per second on a Red Hat Linux machine with 10 CPUs in parallel.

The results are codified in Appendix A. Because the subroutine guarantees success in finding the optimized results, the values of σ_0, as well as the first instance of an optimized schedule and its check matrix can be presented as definitive.

Some interesting results, a few being useful, can be gleaned from Table 1 and Table 2 in Appendix A. They are:

1) Verification of Rule #1 (above). See that the first instance of every optimized schedule (Table 2) begins with day 1 scheduled as meters 1 through *p* placed on piers 1 through *p*. Thus Rule #1 was applied to all other algorithms tested for this paper.
2) It is very common for the *optimal* score to be the *perfect* score (86% of the time for these "small" values of *m*,*p* and *o*)
3) It is *uncommon* for Theorem 4 to predict *all* of the cases when the optimal score is not the perfect score
4) Brute Force is simply too time consuming to compute all possible combinations of *m*, *p* and *o* if *m* or *o* exceed 4.

Brute force is an interesting approach, if only for the purpose of having some definitive values for the cardinality of the sets *E*, E_1, E_2, E_3, as well as being guaranteed finding the optimal schedule. However most modern gravimeter comparisons exceed four meters and/or four observations per meter, and therefore less computationally time consuming approaches were pursued.

One pattern was discerned, though its usefulness is limited, specifically with regard to the cardinality of *E* and E_1. Based on the patterns seen so far, a hypothesis will be posited:

Hypothesis #1: $|E| = |E_1|$ if and only if $m = p$.

It is possible that this hypothesis has a proof, but such a proof was not sought as part of this study, since the application of this hypothesis seems severely limited. In fact, it seems a particularly good thing that $|E| > |E_1|$ seems to be true in most cases. See, for example, the two columns in Table 1 labeled $|E_3|/|E|$ and $|E_3|/|E_1|$. These columns represent the % of solutions found in $|E|$ and found in

[5] All software developed for this study is in the category of "research code" (functional, but not cleaned up and commented or ready for public distribution). One of the many follow-up tasks for this study is to clean up all of the code and distribute it publicly. No matter what language, how functional or how integrated, any code distributed would be located at the NGS website: https://geodesy.noaa.gov.

$|E_1|$ respectively. Note that the percentage of success *dramatically* increases if the search is restricted entirely to $|E_1|$, and not to $|E|$, for those cases when $|E|>|E_1|$.

Additionally, no simple pattern is immediately obvious to determine when $|E_2|=|E_3|$, and no effort was expended in this study to find such a rule. However, it is worth noting that, at least for these small values, no example was found of $|E_3|/|E_2|$ smaller than about 6.7%.

The above two observations lead to the somewhat obvious conclusion that restriction to E_1 or better yet to E_2 will significantly increase the chances that a particular schedule is optimal. If the code necessary to perform those restrictive searches is not onerous, such a restriction should only decrease the total computation time in the search.

However, as a closing reminder of this section, no current method exists (besides Brute force) to guarantee a solution to the GSP. As such, the remainder of these algorithms will seek "best possible schedule within reasonable time constraints".

8.2 Optimized Schedule Search Algorithm #2: Greedy Method

The **greedy method** can basically be described as follows: At every opportunity to make a choice when seeking the optimum solution to a problem, always make the choice which *immediately* adds the most to the success of the problem being sought. The difficulty of the greedy method is that it is short-sighted, missing the bigger picture.

For example, consider the problem of trying to maximize the sum of numbers, by following a decision tree, like in Figure 8 below. The greedy method always looks at the immediate gain from the immediate choice. As such, it would first pick the "+12" choice (over +8) and then the "+14" choice (over +7). That is, it would go down the orange path. But as is clearly evidenced in that figure, to truly find the maximum sum (aka "the optimal solution"), the green path would have been correct. But the greedy method does not look that far ahead.

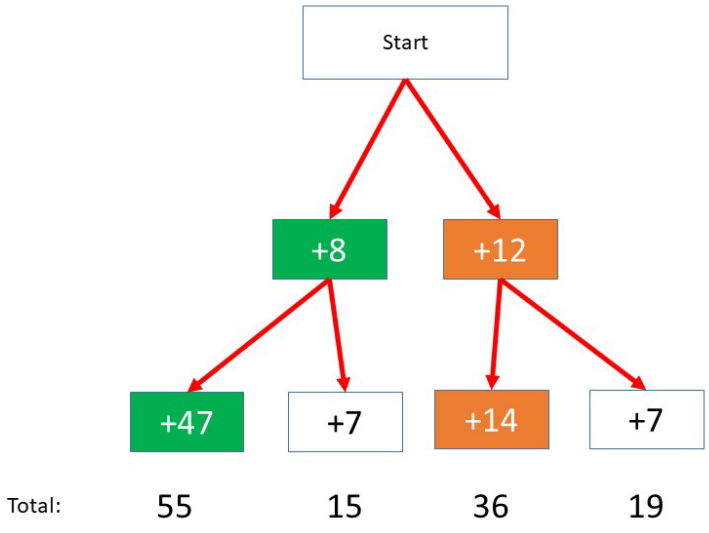

Figure 8: A decision tree for summing numbers, showing the weakness of the greedy method.

Despite this apparent weakness, the greedy method nonetheless has yielded "acceptable" results in scheduling absolute gravimeter comparisons, and is, without a doubt, the fastest method available for any realistic size of gravimeters. By way of example, a schedule for ICAG2017 (28 meters, 9 piers, 4 observations per pier) can be computed using the greedy method in a microsecond.

A computer program called "coags.f" (Comparison of Absolute Gravimeters Scheduler) was developed to apply the greedy method to search for a solution to the GSP for any *m,p,o* combination. It begins by applying Rule #1 (scheduling day 1 as: meters 1->p on piers 1->p), and then it follows the logic below for each day from day 2 through day "d":

1) Make a list of "candidate piers", being every open pier on this day
2) Make a list of "candidate meters", being those meters which:
    a. Have not yet made *o* observations
        i. If every meter has made *o* observations, then allow meters with *o* observations to be considered, so that they might make *o*+1 observations
    b. Have not yet been scheduled on this day
    c. Are not more than 1 observation ahead of all other meters
3) Experimentally schedule each "candidate meter" on every "candidate pier". For each such combination, keep track of how many new meter-to-meter checks are created, and how many self-self checks are created.
4) Make a list of "candidate combinations", being those candidate meters on candidate piers which added zero self-self checks.
    a. If the "candidate combinations" list has no entries (because every combinations yields some self-self checks), then remove the restriction on "zero self-self checks", and put all combinations in the "candidate combinations" list.
5) From the list of candidate combinations, choose the candidate combination which yielded the maximum number of new meter-to-meter checks, and put that candidate meter on that candidate pier.
    a. If there is a tie, randomly choose one candidate combination[6].
6) If this wasn't the last pier of this day, then loop back to #1.
    a. If this was the last pier of the day, then begin investigating the next day, and loop back to #1
        i. If this was the last day, then end program.

It should be noted that the logic listed (specifically #2) above guarantees the schedule must fall within E_1. However nothing in the logic guarantees the schedule falls in E_2.

The greedy method was tested first against small values of *m*,*p* and *o* whose optimal score could be definitively known (from brute force). The results are listed on a column in the first table of Appendix A. Of particular note, the greedy method occasionally finds its way to a schedule with $S > S_0$, putting its solution into set E_1 but not E_2. Curiously this sometimes yields a smaller score than the optimized score. But because minimizing self-self comparisons ($S=S_0$) is part of the solution, such "low score, E_1" schedules are failures of the greedy method.

[6] By randomly choosing, this allows the Greedy Method to be run repeatedly on the same m,p,o combination and yield different answers. This would allow for an "iterative Greedy Method" to be implemented, though such an approach was not pursued for this paper.

Note that the speed of the greedy method is such that it could be set up to iterate as many times as one has patience for. But the successes listed in Table 1 of Appendix A are from a single run. In summary the success rates were:

3 meters: 7 out of 10 (70%)
4 meters: 8 out of 11 (73%)
5 meters: 10 out of 11 (91%)
6 meters: 13 out of 13 (100%)

It is encouraging to see such high success rates. But for values of *m*,*p* or *o* which can't be checked using brute force, it is impossible to know whether the greedy method is achieving the optimal score, unless it achieves the *perfect* score. Therefore there are only two other ways to expand the knowledge of the success of this method: (1) finding the perfect score for larger *m*,*p*, *o* combinations and (2) comparison to the next method, simulated annealing.

8.3 Optimized Schedule Search Algorithm #3: Simulated Annealing

Another approach to finding an optimum solution is to apply a local search algorithm with a cost function thresholding mechanism. This method is frequently used for a variety of NP-complete problems (Aarts and Lenstra 1997; Osogami and Imai, 2000). A local search restricts the problem to a set of candidate solutions which are connected by a series of neighbor-to-neighbor transformations. A cost function threshold directs the search by comparing neighboring candidates and only moving from one to the other if it passes some test. For example, a hill climbing algorithm will only progress to the next candidate solution if its cost function is an improvement upon that of the previous candidate. When the algorithm can no longer find improving moves, it exits and declares the current candidate to be the optimum solution. However, it is possible that a global optimum (our so-called "optimized schedule") may be isolated from many local optima by transformations through poor candidates. This would depend on the defined neighborhood and the means of transforming between neighbors. Hill climbing is unlikely to find a global optimum in such cases.

Simulated annealing is a local search algorithm with thresholding which can overcome the limitations of hill climbing. Like hill climbing, simulated annealing will accept any change which improves the cost function. However, instead of rejecting every change with a high cost, simulated annealing accepts those which pass a stochastic test. This permits the algorithm to explore parts of the search neighborhood which are separated by barriers of high cost. Transformations to schedules of higher cost will be accepted with probability

$$P = e^{-\Delta Cost/T} \tag{27}$$

where *ΔCost* is the change in cost due to the transformation and *T* is a control parameter explained below.

Notice that transformations resulting in small cost increases are more likely to be accepted than large increases. Lower values of *T* enhance the effect. This is an important feature which carries with it the analogies to metallurgy and statistical mechanics. At high temperature *T*, one may transition to any state with almost equal probability. This is the "simmering phase", during which isolated neighborhoods

are easily traversed. As the search progresses, one systematically reduces *T*, resulting in the acceptance of fewer high-cost transformations. Eventually the system will completely stop accepting cost increases and "freeze" into a local optimum just like hill climbing (this "freeze" occurs when the algorithm has tried, and rejected, an unacceptably large number of new schedules. In all simulations in this paper, that rejection limit was 100,000.) However, if *T* is lowered carefully, the system is more likely than hill climbing to find a local optimum that is very close to the global optimum. There are many cooling strategies, but a simple and effective option is to lower it exponentially.

In order to apply simulated annealing to the GSP, it is necessary to define the schedule transformation process:

1) Randomly select any day from the schedule.
2) Randomly select a meter assigned to a pier that day.
3) Randomly select a different meter. This meter may be on a pier or unassigned that day.
4) Swap assignments for the two selected meters. Meters may move between unassigned and assigned.

A computer program called "sags.f" (Simulated Annealing for Gravimeter Scheduling) was developed to apply the simulated annealing method to search for a solution to the GSP for any *m,p,o* combination. The algorithm begins by arbitrarily populating a schedule with physically possible entries (it need not be efficient). The transformation process defines the search neighborhood. If the initial schedule is physically possible, this process will explore only physically possible schedules (e.g. it will never assign one meter to two piers on the same day). In fact, it can be shown that it will explore the entire neighborhood of all physically possible schedules. This is elaborated in Theorem 5.

Theorem 5: Given *m* meters, *p* piers, *d* days, and two arbitrary schedules $J = \{j_{p,d}\}$ and $K = \{k_{p,d}\}$ it is possible to transform schedule *J* into schedule *K* by swapping meters within a given day.

Proof: On day *n*, schedule *J* has meters ordered $\{j_{1,n}, j_{2,n}, \ldots, j_{p,n}, j_{(p+1),n}, \ldots, j_{m,n}\}$, where meters $j_{(p+1),n}$ through $j_{m,n}$ are unassigned on day *n*. To change *J* into *K* on day *n*, swap $j_{i,n}$ with $j_{k_i,n} = k_{i,n}$ from $i = 1$ to *p* until schedule *K* is attained for each day n = 1,2,…,d.

With all physically possible schedules connected, one may therefore begin with any physically possible schedule. Without loss of generality, place meters {1, ..., *p*} on piers {1, ..., *p*}, respectively every day. After that, it is necessary to "simmer" for a short while to obtain a random schedule.

Note that, unless *m*=*p*, this starting schedule is in *E* but not in E_1, since all meters *p*+1 through *m* will have zero observations while all meters *1* through *p* will have *d* observations. Further, note that in the simulated annealing logic listed above, entry #4 allows for the search to continue to reside in *E* rather than E_1. This will be addressed later in the Section 9.

The next step is to define the cost function. Its purpose is to penalize adverse features in the candidate schedules such that its minimum represents the best schedule. This is the aforementioned score (σ) computed by taking the standard deviation of the elements above the diagonal in the check matrix. It is also necessary to penalize schedules with more than the minimum number of self-self checks, S_0, so as to continually guide the search into set E_2, rather than the greater set E_1. This is done by adding 1 to the cost function when the sum of the diagonal is greater than S_0. Another approach might be to scale the penalty by the sum of the diagonal minus S_0, rather than this binary self-self

penalty. However, the binary penalty was ultimately chosen to avoid large jumps in the cost function, which create large barriers that are more difficult for simulated annealing to cross.

Lastly a "Limit" is defined. As each swap is performed, it is tested for whether or not it has improved the schedule. As mentioned earlier, this limit was 100,000 swaps. If the algorithm performs reaches this limit, it ends.

When performed as described above, simulated annealing can quickly and consistently produce perfect schedules (when available) for many problems with large (m, p, o). When perfect schedules are not available, simulated annealing produces "best available" schedules in very short computational times. In Appendix A, example outputs from the algorithm are provided.

In order to investigate whether simulated annealing is always the right choice, a variety of modest (of $m \leq 25$, $p \leq 25$ and $o \leq 12$) combinations were attempted using both the greedy method as well as an increasingly patient version of simulated annealing. The results are tabulated in Appendix B, and conclusions drawn therein. The summary from that appendix is that simulated annealing is, in general, a better choice than the greedy method, but that it was rarely worth the time to push simulated annealing beyond a few minutes of computer time.

9 Discussion and Conclusions

This paper defined the Gravimeter Scheduling Problem (GSP) and its significance in the geodetic world. To take a mathematical approach to solving it, it first laid out the details as an NP-complete problem with an additional optimization constraint. It showed equations for a number of pre-computable statistics which aid in guiding any search for a solution to the GSP. Among them was the so-called "perfect score". Although the perfect score only partially bounds the search, a number of brute force computations showed that the optimal score and the perfect score are identical in 86% of all cases with meters, piers or observations all in the 2 to 6 range. Whether or not that correlation holds for higher values is questionable, as results of higher (6 to 25 range) values of m, p or o generally failed to ever achieve the perfect score.

It was shown that, without a method to pre-compute the optimal score (such as, but not limited to, proving in advance whether the optimal score is equal to the perfect score), all attempts to solve the GSP are limited to one of two choices:

1) Brute force, which guarantees success, if one has the time to wait for every schedule to be checked[7]
2) Optimal search algorithms such as the greedy method or simulated annealing which do not guarantee success, but rather achieve "acceptably good results in an acceptably short period of time"

Brute force is only useful for small values of m, p and o, but is interesting for at least understanding certain patterns in the overall GSP.

[7] A glance at Appendix A shows that many runs of "bruteforce.f" take a few second to minutes, but these times grow out of bounds quickly. The most egregious case where m, p and o are all below 7 would be for (m,p,o) = (6,2,6) which would take approximately 3 billion years to run.

Of the greedy method and simulated annealing, simulated annealing had a significantly more impressive rate of finding a schedule with a smaller score than the greedy method. It was also seen that, in general, it was not worth the time to push simulated annealing beyond a few minutes of computational time, as the results rarely improved.

A substantial number of statistics were generated by many different runs of the three methods, and are captured in Appendices A and B. Some of the more important conclusions drawn are

For small values of *m*, *p* and *o*:

1) Brute force is capable of providing comprehensive insight into all sets E, E_1, E_2 and E_3
2) The chances are high that the optimal score is the perfect score.
3) Both the greedy method and simulated annealing are highly successful, but simulated annealing is clearly superior.
4) The percent of solutions grows significantly when restricting the search to E_1 rather than E
5) The percent of solutions grows significantly when restricting the search to E_2 rather than E_1

For more moderate values of *m*, *p* and *o*:

6) In a random sampling of larger values of *m*, *p* and *o*, neither the greedy method nor simulated annealing found the perfect score raising two questions:
    a. Does the rate of when $\sigma_0 = \sigma_P$ fall off significantly at modest values of *m*, *p* and *o*? and/or
    b. Does the success of the greedy method and simulated annealing fall off at modest values of *m*, *p*, and *o*?
7) The value of S_0 seems to have no impact on success of the greedy algorithm nor simulated annealing.
8) If $\sigma_P = 0$, the chances of success increase dramatically for both the greedy method and simulated annealing

Because each absolute gravimeter comparison event is unique, it is impossible to design it with "just the right number of meters, piers and days" so as to optimize the schedule. Therefore, a simulated annealing computation, run as long as possible in advance, is recommended as the best way to ensure a "near optimal" schedule. Since the Greedy Method always runs in microseconds, and Simulated Annealing rarely takes longer than 5 minutes, there is certainly an advantage to using both methods, since Appendix B shows there are some (rare) cases where the Greedy Method is superior to Simulated Annealing.

10 Future Work

A number of questions arose during this study, though time prevented them from being fully investigated. Any future expansion of this work should consider any or all of the following known improvements and unanswered questions:

Known improvements:

- Change the brute force method to pre-compute *SPD*, and then build schedules by cycling through all *SPD* on each day, rather than going pier by pier.

- Change the greedy method to consider 2 or 3 decisions in the future, rather than 1 at a time
- Change the method of simulated annealing to lay out a schedule within E_1 immediately. Then only swap within the overall schedule (that is, there are no such things as "unassigned meters"). By only re-arranging the schedule as originally laid out, the number of meters making *o* observations and the number of meters making *o*+1 observations will not change, thus guaranteeing that the search will remain within E_1. The only caveat is that a swap must make physical sense (for example a meter must not be assigned to two piers on the same day).

Unanswered questions:

- Can a method be devised to compute σ_0?
- Can a method be devised which predicts when $\sigma_0 = \sigma_P$?
- Can the lessons learned, and code developed in this paper be applied to other geodetic optimization problems?
- How should an additional optimization constraint of "minimize travel costs" be weighed against the other two optimization constraints of "minimize self-self checks" and "balance all other checks"
- How should the distinction between Key Comparison instruments and Pilot Study instruments be implemented within these optimization approaches?
- Can a test be developed which predicts when an optimized schedule also happens to make every possible non-self-self check?

11 Bibliography

Aarts, E., J. Lenstra (1997): Local Search in Combinatorial Optimization. Wiley

Boulanger J. D., Arnautov G. P., Scheglov S. N., Bulletin D'Information, Bureau Gravimetrique International, **52**, June 1983.

Even, S., A. Itai, A. Shamir (1976): On the Complexity of Timetable and Multicommodity Flow problems. *SIAM Journal on Computing*, Vol. 5, No. 4, pp. 691-703

Gunawan, A., K.M. Ng and K.L. Poh (2007): Solving the Teacher Assignment-Course Scheduling Problem by a Hybrid Algorithm. *International Journal of Industrial and Manufacturing Engineering*, Vol. 1, No. 9. pp. 491-496.

Lenz, H., G. Ringel (1991): A brief review on Egmont Köhler's mathematical work. *Discrete Mathematics*, Vol. 97 (1–3), pp. 3–16
Manber, U. (1989): Introduction to Algorithms: A Creative Approach, pp. 341-374 Addison-Wesley Publishing Company

Newell, D.B., D. van Westrum, O. Francis, J. Kanney, J. Liard, A.E. Ramirez, B. Lucero, B. Ellis, F. Greco, A. Pistorio, R. Reudink, D. Iacovone, F. Baccaro, J. Siliker, R. D. Wheeler, R. Falk and A. Ruelke (2017): Regional comparison of absolute gravimeters SIM.M.G-K1 key comparison, *Metrologia*, Vol. 54.

Osogami, T., H. Imai (2000): Classification of Various Neighborhood Operations for the Nurse Scheduling Problem, *IBM TRL Research Report, RT0373*

Pálinkáš, V., O. Francis, M. Val'ko, J. Kostelecký, M. Van Camp, S. Castelein, M. Bilker-Koivula, J. Näränen, A. Lothhammer, R. Falk, M. Schilling, L. Timmen, D. Iacovone, F. Baccaro, A. Germak, E. Biolcati, C. Origlia, F. Greco, A. Pistorio, R. De Plaen, G. Klein, M. Seil, R. Radinovic, R. Reudink, P. Dykowski, M. Sękowski, D. Próchniewicz, R. Szpunar, M. Mojzeš, J. Jańk, J. Papčo, A. Engfeldt, P. A. Olsson, V. Smith, D. van Westrum, B. Ellis and B. Lucero (2017): Regional comparison of absolute gravimeters, EURAMET.M.G-K2 key comparison, *Metrologia*, Vol. 54

Solos, I., I. Tassopoulos, G. Beligiannis (2013): A Generic Two-Phase Stochastic Variable Neighborhood Approach for Effectively Solving the Nurse Rostering Problem. *Algorithms.* Multidisciplinary Digital Publishing Institute. Vol. 6, No. 2, pp. 278–308.

12 Appendix A: Definitive statistics for most $m \leq 6$, $p \leq 6$ and $o \leq 6$ combinations

The brute force method was applied to many combinations of $m \leq 6$, $p \leq 6$ and $o \leq 6$, in order to compute definitive statistics. It was hoped that certain patterns might lead to insights and strategies for future search algorithms. The subroutine *bruteforce.f* was used, capable of checking just under four million schedules per second. As such, certain combinations of *m*,*p* and *o* are estimated as taking too long to actually run, and their entries are incomplete. The results are captured in Table 1 and Table 2.

SPD is the number of schedules per day that are physically possible and unique. $|E|$ is the total number of possible schedules altogether (equation 26). Combinations which take 1 day ($m=p$, $o=1$ => d=1) have been put in red text as uninteresting, as they do not allow any comparisons at all (check matrix is all zeroes). However, as their statistics are part of the overall patterns, these rows have been left in.

The rightmost two columns are the success rates of the greedy method and simulated annealing (with Tstep = 0.9999). Due to a glitch in the logic of the coags.f, the greedy method could not evaluate schedules with very low numbers of observations.

The color coding in Table 1 is done to seek patterns. Yellow tends to mean that two neighboring cells, upon comparison, are different. Green tends to mean that compared neighboring cells are the same.

Table 2 is a companion to Table 1. The entries in Table 2 show just one possible optimized schedule (recall that the entire set E_3 contains *every* optimized schedule, and $|E_3|$ can be quite large; see the Table 1). The *m*,*p*,*o* combinations where the optimized score was not the perfect score are highlighted in yellow, again mostly to look for patterns.

In the columns for Time, the number of seconds to run bruteforce.f is usually given. If the time exceeded a few minutes, then minutes, hours or days might be given. A red box with a time estimate in it indicates the time expected to run bruteforce.f in its current form. Blank red boxes indicate times so large as to be impossible to even attempt.

In the columns for the success of the greedy method and simulated annealing, the following key should be used:

- Green Y = success in achieving optimal score
- Red N = failure to achieve optimal score, with a note (usually showing the achieved score, or showing that the found schedule was not within E_2, etc.)
- Gray σ = The optimal score isn't known, and the method did not achieve the perfect score, but rather that σ shown
- Orange "Logic Failure" = A failure in program "coags.f" to properly deal with the case $o=1$.
- Red "N/A" happens for events *m*,*p*,*o* combinations leading to $d=1$, and thus having no possible checks.

Finally, from a useful/practical standpoint, notice the percentages in column $|\boldsymbol{E_3}|/|\boldsymbol{E_2}|$. These ratios are frequently 100%, but even when they are not, the smallest value so far observed (in the m,p,o < 6 cases) is 6.77%. This means that, happily, a large percentage of schedules that fall in $\boldsymbol{E_2}$ are also optimized schedules. As such, and for future work, it would behoove any seeker of an optimized gravimeter comparison schedule to restrict the schedule search to the $\boldsymbol{E_2}$ domain, as this clearly increases the chance of finding an optimized schedule by a substantial amount over searching across the $\boldsymbol{E_1}$ or $\boldsymbol{E}$ domains. For instance, in Simulated Annealing, the initial schedule can be forced to be in $\boldsymbol{E_2}$, and then by only swapping a meter with an already scheduled meter (while never scheduling one meter on the same day on different piers), it is assured that every newly generated schedule is always in $\boldsymbol{E_2}$.

Table 1: Definitive statistics from application of the brute force method, as well as success rates of the greedy method and simulated annealing, for *m,p,o* values under 7. Yellow indicates unequal numbers in neighboring cells. Peach values in the ratio of cardinalities indicate significantly smaller values than 1%.

m	p	o	d	SPD	|E|	$\|E_1\|$	$\|E_2\|$	$\|E_3\|$	$\frac{\|E_3\|}{\|E\|}$ (%)	$\frac{\|E_3\|}{\|E_1\|}$ (%)	$\frac{\|E_3\|}{\|E_2\|}$ (%)	C	n	S_0	C_0^*	I_{LO}	N_{LO}	I_{HI}	N_{HI}	σ_P	σ_0	Time (s)	Greedy Success	Simulated Annealing Success
2	2	1	1	2	2	2	2	N/A	N/A	N/A	N/A	0	1	0	0	0	1			N/A	N/A	0.00	N/A	N/A
2	2	2	2	2	4	4	2	N/A	N/A	N/A	N/A	2	1	0	2	2	1			N/A	N/A	0.00	N/A	N/A
2	2	3	3	2	8	8	6	N/A	N/A	N/A	N/A	6	1	2	4	4	1			N/A	N/A	0.00	N/A	N/A
2	2	4	4	2	16	16	6	N/A	N/A	N/A	N/A	12	1	4	8	8	1			N/A	N/A	0.00	N/A	N/A
2	2	5	5	2	32	32	20	N/A	N/A	N/A	N/A	20	1	8	12	12	1			N/A	N/A	0.00	N/A	N/A
2	2	6	6	2	64	64	20	N/A	N/A	N/A	N/A	30	1	12	18	18	1			N/A	N/A	0.00	N/A	N/A
3	2	1	2	6	36	24	12	12	33.33	50.00	100.00	2	3	0	2	0	1	1	2	0.58	0.58	0.00	Logic Failure	Y
3	2	2	3	6	216	48	12	12	5.56	25.00	100.00	6	3	0	6	2	3			0	0	0.00	Y	Y
3	2	3	5	6	7,776	2,880	900	900	11.57	31.25	100.00	20	3	4	16	5	2	6	1	0.58	1.16	0.00	Y	N($S \neq S_0$)
3	2	4	6	6	46,656	5,760	900	900	1.93	15.63	100.00	30	3	6	24	8	3			0	0	0.00	N(σ=1.15)	Y
3	2	5	8	6	1,679,616	430,080	94,080	94,080	5.60	21.88	100.00	56	3	14	42	14	3			0	1.73	0.38	Y	N($S \neq S_0$)
3	2	6	9	6	10,077,696	860,160	94,080	94,080	0.93	10.94	100.00	72	3	18	54	18	3			0	0	2.52	N(σ=1.15)	Y
3	3	1	1	6	6	6	6	N/A	N/A	N/A	N/A	0	3	0	0	0	3			N/A	N/A	0.00	N/A	N/A
3	3	2	2	6	36	36	12	12	33.33	33.33	100.00	3	3	0	3	1	3			0	0	0.00	Y	Y
3	3	3	3	6	216	216	12	12	5.56	5.56	100.00	9	3	0	9	3	3			0	0	0.00	Y	Y
3	3	4	4	6	1,296	1,296	216	216	16.66	16.66	100.00	18	3	3	15	5	3			0	0	0.00	Y	Y
3	3	5	5	6	7,776	7,776	900	900	11.57	11.57	100.00	30	3	6	24	8	3			0	0	0.00	Y	Y
3	3	6	6	6	46,656	46,656	900	900	1.93	1.93	100.00	45	3	9	36	12	3			0	0	0.00	N(σ=1.15)	Y
4	2	1	2	12	144	24	24	24	16.66	100.00	100.00	2	6	0	2	0	4	1	2	0.52	0.52	0.00	Logic Failure	N (obs)
4	2	2	4	12	20,736	1,440	216	216	1.04	15.00	100.00	12	6	0	12	2	6			0	0	0.00	Y	Y
4	2	3	6	12	2,985,984	119,040	47,520	47,520	1.59	39.92	100.00	30	6	4	26	4	4	5	2	0.52	0.52	0.00	Y	Y
4	2	4	8	12	429,981,696	11,450,880	748,440	748,440	0.17	6.53	100.00	56	6	8	48	8	6			0	0	97.63	N(σ=0.82)	Y
4	2	5	10	12	61,917,364,224	1,200,697,344	278,510,400	278,510,400	0.45	23.20	100.00	90	6	16	74	12	4	13	2	0.52	0.52	5h8m	N(σ=2.25)	Y
4	2	6	12	12	8,916,100,448,256	133,104,254,976	5,112,676,800	5,112,676,800	0.057	3.84	100.00	132	6	24	108	18	6			0	0	41 d	N($S \neq S_0$)	Y
4	3	1	2	24	576	432	216	144	25.00	33.33	66.67	3	6	0	3	0	3	1	3	0.55	0.55	0.00	Logic Failure	N (obs)
4	3	2	3	24	13,824	5,184	576	576	4.17	11.11	100.00	9	6	0	9	1	3	2	3	0.55	0.55	0.02	Y	Y
4	3	3	4	24	331,776	31,104	576	576	0.17	1.85	100.00	18	6	0	18	3	6			0	0	0.06	Y	Y
4	3	4	6	24	191,102,976	50,388,480	3,032,640	2,397,600	1.25	4.76	79.06	45	6	6	39	6	3	7	3	0.55	1.05	42.96	Y	Y
4	3	5	7	24	4,586,471,424	705,438,720	12,640,320	12,640,320	0.28	1.80	100.00	63	6	9	54	9	6			0	1.10	1231.14	N(σ=1.37)	Y
4	3	6	8	24	110,075,314,176	4,232,632,320	12,640,320	12,640,320	0.0114	2.99	100.00	84	6	12	72	12	6			0	0	9h34m	N($S \neq S_0$)	Y

m	p	o	d	SPD	|E|	$\|E_1\|$	$\|E_2\|$	$\|E_3\|$	$\frac{\|E_3\|}{\|E\|}$ (%)	$\frac{\|E_3\|}{\|E_1\|}$ (%)	$\frac{\|E_3\|}{\|E_2\|}$ (%)	C	n	S_0	C_0^*	I_{LO}	N_{LO}	I_{HI}	N_{HI}	σ_P	σ_0	Time (s)	Greedy Success	Simulated Annealing Success
4	4	1	1	24	24	24	24	N/A	N/A	N/A	N/A	0	6	0	0	0	6			N/A	N/A	0.02	N/A	N/A
4	4	2	2	24	576	576	216	144	25.00	25.00	66.67	4	6	0	4	0	2	1	4	0.52	0.52	0.02	Logic Failure	Y
4	4	3	3	24	13,824	13,824	576	576	4.17	4.17	100.00	12	6	0	12	2	6			0	0	0.02	Y	Y
4	4	4	4	24	331,776	331,776	576	576	0.17	0.17	100.00	24	6	0	24	4	6			0	0	0.09	Y	Y
4	4	5	5	24	7,962,624	7,962,624	83,520	83,520	1.05	1.05	100.00	40	6	4	36	6	6			0	0	2.24	Y	Y
4	4	6	6	24	191,102,976	191,102,976	3,032,640	2,397,600	1.25	1.25	79.06	60	6	8	52	8	2	9	4	0.52	0.52	61.64	N(σ=0.55)	Y
5	2	1	3	20	8,000	1,440	720	720	9.00	50.00	100.00	6	10	0	6	0	4	1	6	0.52	0.52	0.02	Logic Failure	Y
5	2	2	5	20	3,200,000	65,280	5,280	5,280	0.17	8.09	100.00	20	10	0	20	2	10			0	0	0.55	Y	Y
5	2	3	8	20	25,600,000,000	954,777,600	155,433,600	155,433,600	0.61	16.28	100.00	56	10	6	50	5	10			0	0.94	1h43m	N(S≠S_0)	Y
5	2	4	10	20	10,240,000,000,000							90	10	10	80	8	10			0	0*	32.1 d	Y	Y
5	2	5	13	20	81,920,000,000,000,000							156	10	22	134	13	6	14	4	0.52			N(S≠S_0)	σ=1.42
5	2	6	15	20	32,768,000,000,000,000,000							210	10	30	180	18	10			0	0*		N(S≠S_0)	Y
5	3	1	2	60	3,600	1,080	720	720	20.00	66.67	100.00	3	10	0	3	0	7	1	3	0.48	0.48	0.01	Logic Failure	Y
5	3	2	4	60	12,960,000	2,643,840	66,240	66,240	0.51	2.51	100.00	18	10	0	18	1	2	2	8	0.42	0.63	2.35	Y	Y
5	3	3	5	60	777,600,000	15,863,040	66,240	66,240	0.0085	0.42	100.00	30	10	0	30	3	10			0	0	163.44	Y	Y
5	3	4	7	60	2,799,360,000,000	130,506,163,200	3,793,910,400	3,793,910,400	0.136	2.91	100.00	63	10	6	57	5	3	6	7	0.48	0.82	9 days	Y	Y
5	3	5	9	60	10,077,696,000,000,000							108	10	12	96	9	4	10	6	0.52			N(S≠S_0)	σ=1.26
5	3	6	10	60	604,661,760,000,000,000							135	10	15	120	12	10			0	0*		N(S≠S_0)	Y
5	4	1	2	120	14,400	11,520	5,280	3,840	26.67	33.33	72.73	4	10	0	4	0	6	1	4	0.52	0.52	0.02	Logic Failure	Y
5	4	2	3	120	1,728,000	829,440	66,240	57,600	3.33	6.94	86.96	12	10	0	12	1	8	2	2	0.42	0.63	0.39	Y	Y
5	4	3	4	120	207,360,000	39,813,120	161,280	161,280	0.08	0.41	100.00	24	10	0	24	2	6	3	4	0.52	0.52	47.76	Y	Y
5	4	4	5	120	24,883,200,000	955,514,880	161,280	161,280	0.00065	0.017	100.00	40	10	0	40	4	10			0	0*	1h52m	Y	Y
5	4	5	7	120	358,318,080,000,000							84	10	8	76	7	4	8	6	0.52			N(S≠S_0)	σ=0.97
5	4	6	8	120	42,998,169,600,000,000							112	10	12	100	10	10			0			N(S≠S_0)	σ=1.15
5	5	1	1	120	120	120	120	N/A	N/A	N/A	N/A	0	10	0	0	0	10			N/A	N/A	0.02	N/A	N/A
5	5	2	2	120	14,400	14,400	5,280	2,880	20.00	20.00	54.55	5	10	0	5	0	5	1	5	0.53	0.53	0.02	Logic Failure	Y
5	5	3	3	120	1,728,000	1,728,000	66,240	51,840	3.00	3.00	78.26	15	10	0	15	1	5	2	5	0.53	0.53	0.69	Y	Y
5	5	4	4	120	207,360,000	207,360,000	161,280	161,280	0.08	0.08	100.00	30	10	0	30	3	10			0	0	88.47	Y	Y
5	5	5	5	120	24,883,200,000	24,883,200,000	161,280	161,280	0.00065	0.00065	100.00	50	10	0	50	5	10			0	0	3h21m	Y	Y
5	5	6	6	120	2,985,984,000,000							75	10	5	70	7	10			0	0*	9.3 d	Y	Y
6	2	1	3	30	27,000	720	720	720				6	15	0	6	0	9	1	6	0.51	0.51	0.02	Logic Failure	Y
6	2	2	6	30	729,000,000	4,348,800	190,800	190,800	0.026	4.39	100.00	30	15	0	30	2	15			0	0	156.90	Y	Y
6	2	3	9	30	19,683,000,000,000							72	15	6	66	4	9	5	6	0.51	0.51*	61.6 d	Y	Y

m	p	o	d	SPD	|E|	$\|E_1\|$	$\|E_2\|$	$\|E_3\|$	$\frac{\|E_3\|}{\|E\|}$ (%)	$\frac{\|E_3\|}{\|E_1\|}$ (%)	$\frac{\|E_3\|}{\|E_2\|}$ (%)	C	n	S_0	C_0^*	I_{LO}	N_{LO}	I_{HI}	N_{HI}	σ_P	σ_0	Time (s)	Greedy Success	Simulated Annealing Success
6	2	4	12	30	531,441,000,000,000,000							132	15	12	120	8	15			0	0*		$N(S \neq S_0)$	Y
6	2	5	15	30	14,348,907,000,000,000,000,000							210	15	24	186	12	9	13	6	0.51	0.51*		$N(S \neq S_0)$	Y
6	2	6	18	30	387,420,489,000,000,000,000,000,000							306	15	36	270	18	15			0	0*		$N(S \neq S_0)$	Y
6	3	1	2	120	14,400	720	720	720	5.00	100.00	100.00	3	15	0	3	0	12	1	3	0.41	0.41	0.02	Logic Failure	Y
6	3	2	4	120	207,360,000	2,410,560	250,560	250,560	1.21	20.14	100.00	18	15	0	18	1	12	2	3	0.41	0.41	42.51	Y	Y
6	3	3	6	120	2,985,984,000,000	13,866,163,200	15,321,600	15,321,600	0.0005	0.11	100.00	45	15	0	45	3	15			0	0	9.8 d	Y	Y
6	3	4	8	120	42,998,169,600,000,000							84	15	6	78	5	12	6	3	0.41	0.41*		Y	Y
6	3	5	10	120	619,173,642,240,000,000,000							135	15	12	123	8	12	9	3	0.41	0.41*		Y	Y
6	3	6	12	120	8,916,100,448,256,000,000,000,000							198	15	18	180	12	15			0	0*		$N(S \neq S_0)$	Y
6	4	1	2	360	129,600	51,840	30,240	25,920	20.00	50.00	85.71	4	15	0	4	0	11	1	4	0.46	0.46	0.04	Logic Failure	Y
6	4	2	3	360	46,656,000	1,244,160	250,560	86,400	1.85	6.94	34.48	12	15	0	12	0	3	1	12	0.41	0.41	10.03	Logic Failure	Y
6	4	3	5	360	6,046,617,600,000	541,060,300,800	283,046,400	283,046,400	0.0047	0.05	100.00	40	15	0	40	2	5	3	10	0.49	0.62	20.1 d	Y	Y
6	4	4	6	360	2,176,782,336,000,000							60	15	0	60	4	15			0	0*		Y	Y
6	4	5	8	360	282,110,990,745,600,000,000							112	15	8	104	6	1	7	14	0.26			σ=0.88	σ=0.88
6	4	6	9	360	101,559,956,668,416,000,000,000							144	15	12	132	8	3	9	12	0.41	0.41*		$N(S \neq S_0)$	Y
6	5	1	2	720	518,400	432,000	190,800	136,800	26.39	31.67	71.70	5	15	0	5	0	10	1	5	0.49	0.49	0.16	Logic Failure	Y
6	5	2	3	720	373,248,000	207,360,000	15,321,600	3,110,400	0.83	1.50	20.30	15	15	0	15	1	15			0	0.54	113.01	Logic Failure	Y
6	5	3	4	720	268,738,560,000	74,649,600,000	283,046,400	230,169,600	0.086	3.08	81.32	30	15	0	30	2	15			0	0.65	26h1m	Y	Y
6	5	4	5	720	193,491,763,200,000							50	15	0	50	3	10	4	5	0.49	0.49*		Y	Y
6	5	5	6	720	139,314,069,504,000,000							75	15	0	75	5	15			0	0*		Y	Y
6	5	6	8	720	72,220,413,630,873,600,000,000							140	15	10	130	8	5	9	10	0.488			$N(S \neq S_0)$	σ=0.90
6	6	1	1	720	720	720	720	N/A	N/A	N/A	N/A	0	15	0	0	0	15			N/A	N/A	0.02	N/A	N/A
6	6	2	2	720	518,400	518,400	190,800	115,200	22.22	22.22	60.38	6	15	0	6	0	9	1	6	0.51	0.51	0.44	Logic Failure	Y
6	6	3	3	720	373,248,000	373,248,000	15,321,600	1,036,800	0.28	0.28	6.77	18	15	0	18	1	12	2	3	0.41	0.41	309.81	Y	Y
6	6	4	4	720	268,738,560,000	268,738,560,000	283,046,400	207,360,000	0.08	0.08	73.26	36	15	0	36	2	9	3	6	0.51	0.51	67h	Y	Y
6	6	5	5	720	193,491,763,200,000							60	15	0	60	4	15			0	0*		Y	Y
6	6	6	6	720	139,314,069,504,000,000							90	15	0	90	6	15			0	0*		Y	Y

* These optimized values have been confirmed, not through brute force, but because the perfect score was found through either the greedy method or simulated annealing (or both). As noted before, if the perfect score is actually found by a search algorithm, it means the optimized score is the perfect score.

Table 2: The first optimized schedule found, and its check matrix, when using brute force, for select values of *m*,*p* and *o* under 7. Note that yellow rows correspond to those *m*,*p*,*o* combinations where the optimal score has been definitively proven to be different from the perfect score. (This can be visually checked by examining the off-diagonal elements of the check matrices: each such case shows off diagonal entries differing by more than "1" from one another. For example, the {*m*,*p*,*o*}={3,2,3} case shows 6's and 4's (thus differing by 2) in the upper triangular elements of the check matrix. Such check matrices cannot yield a perfect score as per equation 17). Rows in ORANGE are rows whose *m,p,o* combinations with computation times that are too large to have been checked with brute force, but where an optimized score has been found (either through the greedy method or simulated annealing) because the perfect score was found. In these ORANGE rows the schedule and check matrix are not the "first instance found through brute force", but rather "one example, of an unknown number of possible examples, of an optimized schedule, as found through the greedy method or simulated annealing". Rows in RED are rows whose *m,p,o* combinations with computation times that are too large to have been checked with brute force, and where neither the greedy method nor simulated annealing were able to find a perfect schedule, and whose schedules cannot therefore be conclusively stated to be optimized.

m	p	o	d	**First Instance of an Optimized Schedule** (rows are piers 1-p from top to bottom) (columns are days 1-d from left to right) (entries are the meter numbers on that pier/day combination)	**Check Matrix**
3	2	1	2	1 2 2 3	0 1 0 0 0 1 0 0 0
3	2	2	3	1 2 3 2 3 1	0 2 2 0 0 2 0 0 0
3	2	3	5	1 1 2 2 3 2 2 3 3 1	1 6 4 0 2 6 0 0 1
3	2	4	6	1 1 2 2 3 3 2 2 3 3 1 1	2 8 8 0 2 8 0 0 2
3	2	5	8	1 1 1 2 2 2 3 3 2 2 2 3 3 3 1 1	4 15 12 0 6 15 0 0 4
3	2	6	9	1 1 1 2 2 2 3 3 3 2 2 2 3 3 3 1 1 1	6 18 18 0 6 18 0 0 6
3	3	2	2	1 2 2 3 3 1	0 1 1 0 0 1 0 0 0

m	p	o	d	**First Instance of an Optimized Schedule** (rows are piers 1-p from top to bottom) (columns are days 1-d from left to right) (entries are the meter numbers on that pier/day combination)	**Check Matrix**
3	3	3	3	1 2 3 2 3 1 3 1 2	0 3 3 0 0 3 0 0 0
3	3	4	4	1 1 2 3 2 2 3 1 3 3 1 2	1 5 5 0 1 5 0 0 1
3	3	5	5	1 1 2 2 3 2 2 3 3 1 3 3 1 1 2	2 8 8 0 2 8 0 0 2
3	3	6	6	1 1 2 2 3 3 2 2 3 3 1 1 3 3 1 1 2 2	3 12 12 0 3 12 0 0 3
4	2	1	2	1 3 2 4	0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0
4	2	2	4	1 2 3 4 2 1 4 3	0 2 2 2 0 0 2 2 0 0 0 2 0 0 0 0
4	2	3	6	1 1 2 3 3 4 2 2 1 4 4 3	1 4 5 4 0 1 4 5 0 0 1 4 0 0 0 1
4	2	4	8	1 1 2 2 3 3 4 4 2 2 1 1 4 4 3 3	2 8 8 8 0 2 8 8 0 0 2 8 0 0 0 2
4	2	5	10	1 1 1 2 2 3 3 3 4 4 2 2 2 1 1 4 4 4 3 3	4 12 13 12 0 4 12 13 0 0 4 12 0 0 0 4

m	p	o	d	**First Instance of an Optimized Schedule** (rows are piers 1-p from top to bottom) (columns are days 1-d from left to right) (entries are the meter numbers on that pier/day combination)	**Check Matrix**
4	2	6	12	1 1 1 2 2 2 3 3 3 4 4 4 2 2 2 1 1 1 4 4 4 3 3 3	6 18 18 18 0 6 18 18 0 0 6 18 0 0 0 6
4	3	1	2	1 2 2 3 3 4	0 1 0 0 0 0 1 0 0 0 0 1 0 0 0 0
4	3	2	3	1 2 3 2 1 4 3 4 1	0 2 2 2 0 0 1 1 0 0 0 1 0 0 0 0
4	3	3	4	1 2 3 4 2 1 4 3 3 4 1 2	0 3 3 3 0 0 3 3 0 0 0 3 0 0 0 0
4	3	4	6	1 1 2 2 3 4 2 2 1 3 4 3 3 3 4 4 1 2	1 7 6 5 0 2 8 6 0 0 2 7 0 0 0 1
4	3	5	7	1 1 2 2 3 3 4 2 2 1 1 4 4 3 3 3 4 4 1 1 2	3 10 10 10 0 2 8 8 0 0 2 8 0 0 0 2
4	3	6	8	1 1 2 2 3 3 4 4 2 2 1 1 4 4 3 3 3 3 4 4 1 1 2 2	3 12 12 12 0 3 12 12 0 0 3 12 0 0 0 3
4	4	2	2	1 2 2 3 3 4 4 1	0 1 0 1 0 0 1 0 0 0 0 1 0 0 0 0

m	p	o	d	**First Instance of an Optimized Schedule** (rows are piers 1-p from top to bottom) (columns are days 1-d from left to right) (entries are the meter numbers on that pier/day combination)	**Check Matrix**
4	4	3	3	1 2 3 2 1 4 3 4 1 4 3 2	0 2 2 2 0 0 2 2 0 0 0 2 0 0 0 0
4	4	4	4	1 2 3 4 2 1 4 3 3 4 1 2 4 3 2 1	0 4 4 4 0 0 4 4 0 0 0 4 0 0 0 0
4	4	5	5	1 1 2 3 4 2 2 1 4 3 3 3 4 1 2 4 4 3 2 1	1 6 6 6 0 1 6 6 0 0 1 6 0 0 0 1
4	4	6	6	1 1 2 2 3 4 2 2 1 3 4 3 3 3 4 4 1 2 4 4 3 1 2 1	2 9 8 9 0 2 9 8 0 0 2 9 0 0 0 2
5	2	1	3	1 2 4 2 3 5	0 1 0 1 0 0 0 1 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
5	2	2	5	1 2 3 4 5 2 1 4 5 3	0 2 2 2 2 0 0 2 2 2 0 0 0 2 2 0 0 0 0 2 0 0 0 0 0
5	2	3	8	1 1 2 2 3 3 4 5 2 2 1 3 4 5 5 4	1 6 5 4 4 0 2 6 6 6 0 0 1 4 4 0 0 0 1 5 0 0 0 0 1

m	p	o	d	**First Instance of an Optimized Schedule** (rows are piers 1-p from top to bottom) (columns are days 1-d from left to right) (entries are the meter numbers on that pier/day combination)	**Check Matrix**
5	2	4	10	1 1 3 3 4 5 5 2 2 4 2 5 4 4 5 1 3 3 1 2	2 8 8 8 8 0 2 8 8 8 0 0 2 8 8 0 0 0 2 8 0 0 0 0 2
5	2	5	13	Due to time constraints, this hasn't been checked yet.	Due to time constraints, this hasn't been checked yet.
5	2	6	15	1 3 1 1 3 5 4 4 3 4 2 5 2 5 2 2 2 3 5 4 4 5 1 2 3 5 3 1 1 4	6 18 18 18 18 0 6 18 18 18 0 0 6 18 18 0 0 0 6 18 0 0 0 0 6
5	3	1	2	1 2 2 4 3 5	0 1 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0
5	3	2	4	1 2 3 4 2 1 4 5 3 4 5 1	0 2 2 3 2 0 0 1 2 1 0 0 0 2 1 0 0 0 0 2 0 0 0 0 0
5	3	3	5	1 2 3 4 5 2 1 4 5 3 3 4 5 1 2	0 3 3 3 3 0 0 3 3 3 0 0 0 3 3 0 0 0 0 3 0 0 0 0 0
5	3	4	7	1 1 2 2 3 4 5 2 2 1 3 4 5 4 3 3 4 5 5 1 2	1 7 5 5 5 0 2 6 7 6 0 0 1 5 6 0 0 0 1 5 0 0 0 0 1
5	3	5	9	Due to time constraints, this hasn't been checked yet.	Due to time constraints, this hasn't been checked yet.

m	p	o	d	**First Instance of an Optimized Schedule** (rows are piers 1-p from top to bottom) (columns are days 1-d from left to right) (entries are the meter numbers on that pier/day combination)	**Check Matrix**
5	3	6	10	1 5 4 5 2 4 3 1 2 3 2 2 3 4 5 3 1 5 1 4 3 4 5 1 1 2 2 3 4 5	3 12 12 12 12 0 3 12 12 12 0 0 3 12 12 0 0 0 3 12 0 0 0 0 3
5	4	1	2	1 2 2 3 3 1 4 5	0 1 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0
5	4	2	3	1 2 3 2 1 4 3 4 5 4 5 1	0 2 1 2 1 0 0 1 1 0 0 0 0 1 1 0 0 0 0 2 0 0 0 0 0
5	4	3	4	1 2 3 4 2 1 4 5 3 4 5 2 4 5 1 3	0 2 2 3 2 0 0 2 3 2 0 0 0 3 2 0 0 0 0 3 0 0 0 0 0
5	4	4	5	1 2 3 4 5 2 1 4 5 3 3 4 5 2 1 4 5 1 3 2	0 4 4 4 4 0 0 4 4 4 0 0 0 4 4 0 0 0 0 4 0 0 0 0 0
5	4	5	7	Due to time constraints, this hasn't been checked yet.	Due to time constraints, this hasn't been checked yet.
5	4	6	8	Due to time constraints, this hasn't been checked yet.	Due to time constraints, this hasn't been checked yet.
5	5	2	2	1 2 2 3 3 4 4 5 5 1	0 1 0 0 1 0 0 1 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0

m	p	o	d	**First Instance of an Optimized Schedule** (rows are piers 1-p from top to bottom) (columns are days 1-d from left to right) (entries are the meter numbers on that pier/day combination)	**Check Matrix**
5	5	3	3	1 2 3 2 1 4 3 4 5 4 5 1 5 3 2	0 2 1 2 1 0 0 2 1 1 0 0 0 1 2 0 0 0 0 2 0 0 0 0 0
5	5	4	4	1 2 3 4 2 1 4 5 3 4 5 2 4 5 1 3 5 3 2 1	0 3 3 3 3 0 0 3 3 3 0 0 0 3 3 0 0 0 0 3 0 0 0 0 0
5	5	5	5	1 2 3 4 5 2 1 4 5 3 3 4 5 2 1 4 5 1 3 2 5 3 2 1 4	0 5 5 5 5 0 0 5 5 5 0 0 0 5 5 0 0 0 0 5 0 0 0 0 0
5	5	6	6	1 5 3 4 4 2 2 4 5 1 3 5 3 2 4 2 5 1 4 3 1 5 2 3 5 1 2 3 1 4	1 7 7 7 7 0 1 7 7 7 0 0 1 7 7 0 0 0 1 7 0 0 0 0 1
6	2	1	3	1 3 5 2 4 6	0 0 1 0 1 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
6	2	2	6	1 2 3 4 5 6 2 1 4 3 6 5	0 2 2 2 2 2 0 0 2 2 2 2 0 0 0 2 2 2 0 0 0 0 2 2 0 0 0 0 0 2 0 0 0 0 0 0

m	p	o	d	**First Instance of an Optimized Schedule** (rows are piers 1-p from top to bottom) (columns are days 1-d from left to right) (entries are the meter numbers on that pier/day combination)	**Check Matrix**
6	2	3	9	1 5 3 3 2 5 6 6 4 2 2 1 6 1 4 4 5 3	1 5 4 5 4 4 0 1 4 5 4 4 0 0 1 4 5 5 0 0 0 1 4 4 0 0 0 0 1 5 0 0 0 0 0 1
6	2	4	12	1 5 1 4 3 5 2 6 4 6 3 2 2 1 5 3 2 1 6 3 5 4 4 6	2 8 8 8 8 8 0 2 8 8 8 8 0 0 2 8 8 8 0 0 0 2 8 8 0 0 0 0 2 8 0 0 0 0 0 2
6	2	5	15	1 4 6 3 1 2 6 4 6 3 5 2 3 1 5 2 3 5 1 6 5 4 2 1 5 2 4 6 4 3	4 12 13 12 12 13 0 4 12 13 13 12 0 0 4 12 12 13 0 0 0 4 13 12 0 0 0 0 4 12 0 0 0 0 0 4
6	2	6	18	1 3 5 3 1 6 2 4 6 4 1 2 6 3 4 2 5 1 2 4 3 2 3 1 5 6 1 6 4 5 2 4 5 3 6 1	6 18 18 18 18 18 0 6 18 18 18 18 0 0 6 18 18 18 0 0 0 6 18 18 0 0 0 0 6 18 0 0 0 0 0 6
6	3	1	2	1 4 2 5 3 6	0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

m	p	o	d	**First Instance of an Optimized Schedule** (rows are piers 1-p from top to bottom) (columns are days 1-d from left to right) (entries are the meter numbers on that pier/day combination)	**Check Matrix**
6	3	2	4	1 2 3 4 2 1 5 6 3 4 6 5	0 2 1 1 1 1 0 0 1 1 1 1 0 0 0 2 1 1 0 0 0 0 1 1 0 0 0 0 0 2 0 0 0 0 0 0
6	3	3	6	1 2 3 4 5 6 2 1 4 5 6 3 3 4 1 6 2 5	0 3 3 3 3 3 0 0 3 3 3 3 0 0 0 3 3 3 0 0 0 0 3 3 0 0 0 0 0 3 0 0 0 0 0 0
6	3	4	8	1 6 5 2 4 3 3 5 2 1 3 4 5 4 6 6 3 4 6 5 2 2 1 1	1 6 5 5 5 5 0 1 5 5 5 5 0 0 1 5 6 5 0 0 0 1 5 6 0 0 0 0 1 5 0 0 0 0 0 1
6	3	5	10	1 3 6 4 2 6 5 2 3 1 2 6 2 3 1 4 6 5 4 5 3 5 1 2 4 5 4 3 1 6	2 8 9 8 8 8 0 2 8 8 8 9 0 0 2 8 8 8 0 0 0 2 9 8 0 0 0 0 2 8 0 0 0 0 0 2
6	3	6	12	1 2 5 1 5 6 3 4 3 2 6 4 2 5 1 4 6 5 4 1 6 3 3 2 3 6 4 3 1 2 5 6 2 1 4 5	3 12 12 12 12 12 0 3 12 12 12 12 0 0 3 12 12 12 0 0 0 3 12 12 0 0 0 0 3 12 0 0 0 0 0 3

m	p	o	d	**First Instance of an Optimized Schedule** (rows are piers 1-p from top to bottom) (columns are days 1-d from left to right) (entries are the meter numbers on that pier/day combination)	**Check Matrix**
6	4	1	2	1 2 2 3 3 5 4 6	0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0
6	4	2	3	1 2 4 2 3 6 3 5 1 4 6 5	0 1 1 1 1 0 0 0 1 1 0 1 0 0 0 0 1 1 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0
6	4	3	5	1 2 3 4 5 2 1 4 3 6 3 4 5 6 1 4 3 6 5 2	0 2 3 3 2 2 0 0 3 3 2 2 0 0 0 4 3 3 0 0 0 0 3 3 0 0 0 0 0 2 0 0 0 0 0 0
6	4	4	6	1 4 2 6 5 3 2 6 3 5 1 4 3 1 5 2 4 6 4 5 6 1 3 2	0 4 4 4 4 4 0 0 4 4 4 4 0 0 0 4 4 4 0 0 0 0 4 4 0 0 0 0 0 4 0 0 0 0 0 0
6	4	5	8	Due to time constraints, this hasn't been checked yet.	Due to time constraints, this hasn't been checked yet.
6	4	6	9	1 5 4 6 2 3 2 1 5 2 3 6 2 6 5 3 4 1 3 2 3 4 1 4 1 5 6 4 4 5 1 5 2 6 6 3	2 9 9 9 9 8 0 2 9 8 9 9 0 0 2 9 8 9 0 0 0 2 9 9 0 0 0 0 2 9 0 0 0 0 0 2

m	p	o	d	**First Instance of an Optimized Schedule** (rows are piers 1-p from top to bottom) (columns are days 1-d from left to right) (entries are the meter numbers on that pier/day combination)	**Check Matrix**
6	5	1	2	1 2 2 3 3 1 4 5 5 6	0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0
6	5	2	3	1 2 3 2 1 4 3 4 5 4 3 6 5 6 1	0 2 1 1 1 1 0 0 1 1 0 0 0 0 0 2 1 1 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0
6	5	3	4	1 2 3 4 2 1 4 5 3 4 5 6 4 3 6 1 5 6 1 2	0 3 2 3 2 2 0 0 1 2 2 1 0 0 0 3 1 2 0 0 0 0 2 2 0 0 0 0 0 2 0 0 0 0 0 0
6	5	4	5	1 6 4 2 3 2 5 1 6 4 3 1 6 5 2 4 2 5 3 1 5 4 3 1 6	0 4 4 4 4 4 0 0 3 3 3 3 0 0 0 3 3 3 0 0 0 0 3 3 0 0 0 0 0 3 0 0 0 0 0 0
6	5	5	6	1 6 2 3 4 5 2 3 6 5 1 4 3 4 5 1 2 6 4 5 1 2 6 3 5 1 4 6 3 2	0 5 5 5 5 5 0 0 5 5 5 5 0 0 0 5 5 5 0 0 0 0 5 5 0 0 0 0 0 5 0 0 0 0 0 0
6	5	6	8	Due to time constraints, this hasn't been checked yet.	Due to time constraints, this hasn't been checked yet.

m	p	o	d	**First Instance of an Optimized Schedule** (rows are piers 1-p from top to bottom) (columns are days 1-d from left to right) (entries are the meter numbers on that pier/day combination)	**Check Matrix**
6	6	2	2	1 2 2 3 3 1 4 5 5 6 6 4	0 1 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 1 0 0 0 0 0 0
6	6	3	3	1 2 3 2 1 4 3 4 5 4 3 6 5 6 1 6 5 2	0 2 1 1 1 1 0 0 1 1 1 1 0 0 0 2 1 1 0 0 0 0 1 1 0 0 0 0 0 2 0 0 0 0 0 0
6	6	4	4	1 2 3 4 2 1 4 5 3 4 5 6 4 3 6 1 5 6 1 2 6 5 2 3	0 3 2 3 2 2 0 0 2 2 3 2 0 0 0 3 2 3 0 0 0 0 2 2 0 0 0 0 0 3 0 0 0 0 0 0
6	6	5	5	1 3 5 2 6 2 5 4 1 3 3 4 1 6 5 4 2 6 3 1 5 6 3 4 2 6 1 2 5 4	0 4 4 4 4 4 0 0 4 4 4 4 0 0 0 4 4 4 0 0 0 0 4 4 0 0 0 0 0 4 0 0 0 0 0 0
6	6	6	6	1 2 6 4 3 5 2 6 4 1 5 3 3 1 2 5 4 6 4 5 3 2 6 1 5 3 1 6 2 4 6 4 5 3 1 2	0 6 6 6 6 6 0 0 6 6 6 6 0 0 0 6 6 6 0 0 0 0 6 6 0 0 0 0 0 6 0 0 0 0 0 0

13 Appendix B: Perfect Scores, and the ability of Greedy Method and Simulated Annealing to achieve them for larger *m*,*p* and *o*.

Recall the following facts about combinations of $m \le 6$, $p \le 6$ and $o \le 6$ from Appendix A:

1) It is very common for the *optimal* score to be the *perfect* score. For those which could be checked (see Table 1), $\sigma_0 = \sigma_P$ occurred in 50 of 58 confirmed cases (86%).
2) The greedy method and simulated annealing had a success rate, in general, between 70% and 100%, for many combinations of $m \le 6$, $p \le 6$ and $o \le 6$, with simulated annealing being noticeably superior.

It was therefore hypothesized that the greedy method and/or simulated annealing might have a fairly high success rate of achieving the perfect score for higher values of *m*,*p* and *o*. To test this, a pseudo-randomly[8] determined set of 30 different *m,p,o* combinations, ranging from $10 \le m \le 25$, $10 \le p \le m$ and $4 \le o \le 8$ each were chosen for testing. For each combination, the defining value to be in set E_2 ($S = S_0$) was computed, and the perfect score (σ_P) as well. These were then used as a test dataset for both the greedy method and simulated annealing. The results are in Table 3, and discussed below it. If a failure occurs within set E_2, then only the failing score is noted. If the failure occurs because a schedule falls outside of E_2, that will also be noted.

The color coding below is as follows: If the method yielded the perfect (and thus optimized) schedule, it is in green and the time it took is noted. If no method yielded the perfect schedule, then the best of the four (greedy, and 3 types of simulated annealing) is coded in yellow. All other solutions for that *m,p,o* combination are coded in red.

Table 3: A sampling of combinations of larger *m,p,o* and their S_0 and σ_P values, as well as the success of the greedy method and simulated annealing to achieve the perfect score

m	p	o	d	S_0	σ_P	Greedy Method Success	Simulated Annealing Success (TStep=0.99)	Simulated Annealing Success (TStep=0.999)	Simulated Annealing Success (TStep=0.9999)
10	10	6	6	0	0.48	N (σ=0.74)	Y 0:00.02	Y 0:00.02	Y 0:00.06
10	10	7	7	0	0.48	N (σ=0.60)	Y 0:00.02	Y 0:00.04	Y 0:00.17
11	10	7	8	0	0.29	N (σ=0.78)	N (σ=0.67), 0:02	N (σ=0.67), 0:10	N (σ=0.67), 3:55
11	11	7	7	0	0.40	N (σ=0.87)	N (σ=0.45), 0:04	Y 0:00.04	Y 0:00.43
12	10	7	9	0	0.50	N (σ=0.79)	N (σ=0.73), 0:04	N (σ=0.73), 0:20	N (σ=0.73), 2:11
13	12	4	5	0	0.50	N (σ=0.75)	N (σ=0.55), 0:04	N (σ=0.55), 0:26	N (σ=0.55), 4:06
13	11	5	6	0	0.32	N (σ=0.82)	N (σ=0.58), 0:03	N (σ=0.58), 0:29	N (σ=0.58), 2:55
14	11	8	11	0	0.48	N (σ=0.74)	N (σ=0.72), 0:03	N (σ=0.72), 0:22	N (σ=0.72), 3:20
14	13	4	5	0	0.50	N (σ=0.63)	N (σ=0.56), 0:03	N (σ=0.56), 0:33	N (σ=0.56), 3:04
14	13	8	9	0	0.35	N (σ=0.85)	N (σ=0.68), 0:21	N (σ=0.68), 3:10	N (σ=0.68), 11:13
15	10	6	9	0	0.50	N (σ=0.85)	N (σ=0.53), 0:09	N (σ=0.53), 0:42	Y 0:01
15	12	6	8	0	0.40	N (σ=0.88)	N (σ=0.63), 0:17	N (σ=0.60), 0:51	N (σ=0.58), 11:02

[8] Using the built in random number generator of Microsoft Excel 2016. That function was not particularly satisfactory, yielding 3 identical *m,p,o* combinations out of 30 when first called. As such, duplicates were removed and replaced with hand-picked combinations. Thus the term “pseudo random” is used.

16	10	8	13	0	0.50	N (σ=0.58)	N (σ=0.58), 0:03	N (σ=0.58), 0:11	N (σ=0.58), 1:26
16	14	8	10	0	0.44	N (σ=0.95)	N (σ=0.61), 0:21	N (σ=0.58), 3:07	N (σ=0.58), 0:17
18	10	8	15	0	0.35	N (σ=0.69)	N (σ=0.69), 0:08	N (σ=0.69), 1:11	N (σ=0.69), 9:01
18	16	6	7	0	0.40	N (σ=0.87)	N (σ=0.60), 0:12	N (σ=0.60), 0:25	N (σ=0.60), 6:17
18	18	5	5	0	0.38	N (σ=0.56)	N (σ=0.46), 0:02	N (σ=0.49), 0:32	N (σ=0.42), 14:03
19	11	5	9	0	0.47	N (σ=0.86)	N (σ=0.68), 0:02	N (σ=0.68), 1:33	N (σ=0.68), 22:48
19	12	5	8	0	0.19	N (σ=0.81)	N (σ=0.62), 0:27	N (σ=0.62), 2:09	N (σ=0.61), 43:10
20	11	4	8	0	0.49	N (σ=0.65)	N (σ=0.68), 0:11	N (σ=0.68), 1:14	N (σ=0.68), 21:04
20	11	5	10	0	0.49	N (σ=0.88)	N (σ=0.75), 0:02	N (σ=0.75), 2:13	N (σ=0.75), 0:40
20	11	7	13	0	0.50	N (σ=0.85)	N (σ=0.66), 0:03	N (σ=0.65), 0:50	N (σ=0.65), 28:54
21	14	5	8	0	0.34	N (σ=0.86)	N (σ=0.67), 0:38	N (σ=0.67), 5:56	N (σ=0.67), 17:32
22	17	8	11	0	0.21	N (σ=1.01)	N (σ=0.73), 1:25	N (σ=0.71), 18:07	N (σ=0.71), 14:39
23	14	6	10	0	0.50	N (σ=0.94)	N (σ=0.69), 0:03	N (σ=0.68), 3:01	N (σ=0.68), 1:35
23	16	7	11	0	0.50	N (σ=1.01)	N (σ=0.73), 0:33	N (σ=0.72), 0:10	N (σ=0.71), 24:49
25	13	6	12	0	0.35	N (σ=0.97)	N (σ=0.76), 0:43	N (σ=0.76), 8:45	N (σ=0.76), 5:01
25	18	8	12	0	0.20	N (σ=1.11)	N (σ=0.77), 1:16	N (σ=0.77), 9:22	N (σ=0.77), 11:16
25	19	5	7	0	0.47	N (σ=0.72)	$N(S \neq S_0)$	N (σ=0.68), 11:23	N (σ=0.76), 1:44:39
25	20	7	9	0	0.49	N (σ=0.94)	N (σ=0.68), 0:04	N (σ=0.65), 1:09	N (σ=0.63), 1:34:39

The generally worse behavior of the greedy method, relative to simulated annealing, is not unexpected. However the generally close clustering of scores, from perfect, to simulated annealing to greedy indicates that the greedy method is not a terrible choice, especially considering its speed. Of further note, simulated annealing tends to have identical or nearly identical results a large proportion of the time, without regard to the Tstep value chosen. As such, unless every last bit of optimization is needed, it seems unnecessary to run simulated annealing with a high Tstep and thus save on minutes of computation time.

The extremely low success rate of both the greedy method and simulated annealing to achieve the perfect score is discouraging at first glance. However, expectations of success are hinged upon two unproven hypotheses:

1) That the success rate (70-100%) of the greedy method and simulated annealing in the low values of *m*,*p* and *o* (under 6 or so) would hold for higher values
2) That the percent of time the optimal score is the perfect score (86%) would continue to be high

It is also interesting that simulated annealing occasionally succeeds in finding the perfect score, but that success rate drops off as the size of *m* grows.

One other oddity was noted in the above table. The 30 pseudo-randomly selected values of *m*, *p* and *o* always yielded examples where $S_0=0$ and $\sigma_P \neq 0$. This was somewhat unexpected, since a review of Table 1 shows that, for smaller *m,p,o* combinations this situation only arose 29% (24 out of 84) of the time. However, a closer look shows that the incidence of this

combination steadily rises as the number of meters increases (8% for 3 meters, 22% for 4, 33% for 5 and 36% for 6). A quick computational check confirmed that the incidence of S_0=0 and σ_P≠0 occurring together rises steadily with the number of meters. The table below shows this for the more expanded possibilities of 2≤*m*≤25, 1≤*p*≤m and 1≤*o*≤12

Table 4: Percent of the time that S_0=0 and σ_P≠0 occurs, by number of meters (for 2≤m≤25, 1≤p≤m and 1≤o≤12)

m	%	m	%	m	%
2	0.00%	10	34.17%	18	62.50%
3	2.78%	11	38.64%	19	64.47%
4	8.33%	12	43.06%	20	66.67%
5	13.33%	13	47.44%	21	65.87%
6	15.28%	14	51.79%	22	68.94%
7	20.24%	15	53.89%	23	70.65%
8	27.08%	16	56.77%	24	72.22%
9	31.48%	17	61.27%	25	72.67%

In order to better understand how the specific cases of S_0≠0, σ_P=0, or both influence the statistics in Table 3, some specific examples were hand-picked and checked. The results are below in Table 5.

Table 5: A sampling of combinations of larger *m,p,o* and their S_0 and σ_P values, as well as the success of the greedy method and simulated annealing to achieve the perfect score, for the specific cases of S_0≠0 , σ_P=0 or both.

m	p	o	d	S_0	σ_P	Greedy Method Success	Simulated Annealing Success (TStep=0.99, Limit = 100,000)	Simulated Annealing Success (TStep=0.999, Limit = 100,000)	Simulated Annealing Success (TStep=0.9999, Limit = 100,000)
12	6	6	12	0	0.00	Y	Y 0:00.04	Y 0:00.05	Y 0:00.16
14	7	7	14	0	0.00	Y	Y 0:00.04	Y 0:00.10	Y 0:00.17
8	8	9	9	8	0.00	Y	Y 0:00.04	Y 0:02.55	Y 0:17.14
8	4	8	16	32	0.00	N (σ=1.93), **S=46**	Y 0:02.14	Y 0:03.82	Y 0:28.77
8	9	12	11	27	0.46	N (σ=1.40), **S=34**	N (σ=0.46), **S=32**	N (σ=0.46), **S=32**	N (σ=0.46), **S=32**
10	11	12	11	11	0.40	N (σ=0.40), **S=22**	N (σ=0.42), **S=20**	N (σ=0.42), **S=20**	N (σ=0.42), **S=20**

Two things were noted from Table 5:

1) When the perfect score is zero, both the greedy method and simulated annealing succeed at finding it at a much higher rate than in other cases.
2) When S_0 is not zero, the greedy method and simulated annealing can sometimes succeed (possibly helped by #2, above), but in other cases they tend toward a solution which isn't even in E_2. In the case of simulated annealing, this could be an indication that the "penalty" for adding self-self checks is not properly calibrated for cases where a particular (non-zero) of S_0 is actually the target.

14 Appendix C: Current practice versus "proposed" practice

All of the optimization work in this paper relied upon a defined way to run a comparison of absolute gravimeters. To summarize:

1) Set the "end trigger" to be: each meter has made at least "*o*" observations
2) Never leave a pier empty
3) Minimize the number of days it takes to accomplish the event

Under these three rules, further rules are applied, being:

4) Allow meters to make o+1 observations, but no more than that
5) Minimize self-self checks, but do not break rule #2 in an attempt to force them to zero
6) Allow the number of checks between two meters to be zero if (based on rules 1-3) it is mathematically impossible to do so given the *m*,*p*,*o* values.

In this case, the definition of an "optimized schedule" is one which fulfills these two criteria:

a) $S=S_0$
b) $\sigma=\sigma_0$

Let us refer to the above rule set as "**proposed practice**". The authors fully recognize that this set of rules differs from how comparisons are commonly designed at the present. Let us summarize the "rules" of what we will call "**current practice**" below. Note that we are assuming that all instruments are of "equal importance" in a comparison. That is, we are ignoring the distinction between Key Comparison instruments and Pilot Study instruments which usually occurs at comparisons. That is yet another refinement for future study.

1) Set the "end trigger" to be: each meter has made *exactly* "*o*" observations
2) Never let a self-self check occur

Under these two rules, further rules are applied, being:

3) Any pier may sit vacant on any given day
4) While keeping the duration (d) of the event "reasonable", try to have each meter make a check against every other meter at least once

In this case, the definition of an "optimized schedule" is one which fulfills these two criteria:

a) S=0
b) $\sigma=\sigma_0$
c) (If possible) every non-diagonal element in the check matrix is non-zero

Note that a balanced set of non-self-self checks ($\sigma=\sigma_0$) is the same in both current practice and proposed practice, but the adherence to *S*=0 in current practice as well as the adherence that all meters make an identical number of observations yields some inefficiencies from which we hope to dissuade future schedulers.

The primary argument for switching to proposed practice derives from the opening part of the paper: gravimeter comparisons are *expensive*. Much of this argument comes down to one thing: letting a pier sit idle is costly with no associated gain.

Having a pier sit idle while a meter could be taking measurements is wasteful. The operator is expending travel costs for a day in order to sit idle. We accept that this must happen in any event when *m*>*p*, but to exacerbate this fact with purposeful idleness is not defensible. We make the supposition that having extra measurements enhances the ability to analyze the comparison as a whole. Specifically,

consider the situation where some meter takes observations only to find out later that they were corrupted in some way. Such observations would need to be removed from the analysis. We argue two ways here: First, that having the possibility of one extra measurement for some meters gives the possibility that the extra measurement can stand in for a corrupt measurement. Secondly, if having the exact same number of observations is absolutely essential to the analysis, then every time a corrupt observation is taken and must be dropped, then the analysis is made sub-standard through imbalance anyway. We argue, with years of experience, that rarely (if ever) has an absolute gravimeter comparison occurred in which at least one observation was not questioned or discarded. Therefore if an imbalance in the number of observations, a-posteriori, is going to happen anyway, why not embrace it and allow for extra measurements and fully occupied piers on every day?

There is an anecdotal feeling among current schedulers that it is better for a pier to sit empty than for a self-self check to occur, or for a meter to have more observations than other meters. However some simple tests designed explicitly to test this hypothesis show that self-self checks do not "bias toward the instrument" that had a self-self check, nor does having an additional measurement cause any biasing toward that instrument. However a comprehensive study on this has not been performed, to the knowledge of the authors[9]. We argue that taking extra measurements, at the very least, does no harm. Certainly the extra measurement could be *excluded* from the final least squares adjustment if necessary.

However in large comparisons where common practice is to break the schedule into sets of instruments, there is the potential for the last set to be smaller than all preceding sets, thus causing vacant piers. This sort of vacancy is not at issue. Rather we argue against purposefully idle meters when they are on hand, operational, with an available vacant pier, yet not taking measurements that day.

In summary, one can always *exclude* observations that have been taken, but one cannot *add* observations that were purposefully skipped.

14.1 A numerical example

We examine current practice and proposed practice under the light of a numerical example. Consider the case of (*m*,*p*) = (10,7). Under proposed practice, the trigger would be to define "*o*". Under current practice the trigger would be to define "o" as well, but tempered with an attempt to accomplish as many potential checks as possible, while avoiding self-self checks (S=0).

While current practice does not always force a check between all meters, such a target is viewed as worthwhile, provided the number of days is not prohibitive. Let us begin by examining whether it is possible to compute how many days it would take to make every potential check. With 10 meters, the number of potential checks is *n*=45. The total number of checks performed over the course of the event is found in equation 4 as:

$$C = p\binom{d}{2} = \frac{pd(d-1)}{2}$$

We know there are 7 piers, but we do not know how many days the event lasts, so we set up this equation to determine it:

[9] However it is underway at NGS as of 2019 (Smith and van Westrum, personal communication)

$$45 = \frac{7d(d-1)}{2}$$

Solving for *d* tells us that the event must last at least 5 days, for 10 meters on 7 piers to make at least 45 checks. Note that this doesn't guarantee that every one of the 45 potential checks is made; it just guarantees that at least 45 checks are made. Some might be duplicates. If no pier sits vacant (against current practice), then over 5 days the total number of checks made on 7 piers is 70 (this would have been 42 over only 4 days, according to equation 4).

Let us now try to design a schedule which enforces S=0 and which makes every potential check (no zero off-diagonal elements in the check matrix.) As we do not know how long this will take, we must begin with the minimum duration, as derived above, and assume *d*=5. With 5 days and 7 piers, this means a total of 35 observations might be taken. However, current practice rule #3 says all meters must make *the exact same number of observations*. For this to happen, the maximum number of observations possible is 3 per meter for a total of 30 observations. Thus, over the course of a (minimum) 5 day event, there will be 5 times when a pier sits vacant. This is one of the inefficiencies of current practice we wish to avoid, as explained below.

When a pier sits vacant it reduces the overall number of checks which occur on that pier and thus in an event. Let us say, for the sake of argument and just to grasp the magnitude of the issue, that over the 5 day event that piers 1 through 5 sit vacant on days 1 through 5 (leaving 30 observations still in the 5 day schedule, during which all 10 meters will make exactly 3 observations). This means that on piers 1-5 the number of checks which occur is the same as if the event had been only 4 days long. That is, on piers 1-5 the number of checks drops from 10 (5 × 4 / 2) down to 6 (4 × 3 / 2). Piers 6 and 7 retain their full complement of 10 checks. Thus the total number of checks over 5 days has reduced down from 70 to 50. This remains above the desired 45, and so it seems (at least mathematically) *possible* to design a 5 day schedule under current practice with 50 checks.

We stop here temporarily and briefly step down the rabbit hole of ridiculousness, just to make a point: We note that 50 is not 45, and therefore (in the above example) even if every potential check has been made there will be 5 checks between meters which occur twice, while the other 40 potential checks will occur only once. If perfect balance in number of *observations* is so important to current practice, we argue that one might conclude that perfect balance in *checks* is too, and thus one might set the goal that the score of any schedule should be zero. If that belief is followed, such an imbalance in checks might be viewed as "skewing" the results toward the meters which participated in the extra checks. If such skewing is seen as "bad", then somehow the schedule must be designed to have the number of checks be exactly 45. That is, we must somehow force $C^* = n$ (or some integer multiple of n), and thus this exercise of finding a schedule must continue by modifying again and again, leaving piers empty, adding more days, and attempting to find such a "non skewed" schedule where *both* observations *and* checks are perfectly balanced. If such skewing is not seen as a problem, then the question must be raised: wouldn't it be better to have 70 checks than 50? If the answer is "yes", then piers should not sit vacant. We conclude that attempting to balance checks adds a further constraint to the GSP which is unnecessary and somewhat ridiculous.

We gingerly remove ourselves from that rabbit hole and continue to analyze our example. If pier vacancy is done solely to avoid self-self checks, then consider the following: applying equation 19 to likely event scenarios ($m<30$, $p<20$, $o<10$) implies that S_0 is non-zero (self-self checks are unavoidable) 22% of the time (under *proposed* practice). Of those times, they are heavily skewed to combinations of *m,p,o* where $p<<m$; when this happens, but even then it takes larger and larger values of *o* for this to occur. Thus, even

for radically off balance cases (m=20, p=3, o=3), schedules can be designed with S_0=0. (In this specific case, not only is S_0=0, but the optimal schedule is a perfect schedule with a score of 0, all without any pier vacancies!). The point is this: artificially leaving piers vacant in order to force S=0 is not even necessary 78% of the time, and in the other times, we hope the above arguments push current practice toward the proposed practice of this paper.

How does proposed practice apply to, say, the ICAG17 from China with 28 meters, 9 piers, and 4 observations? There were 28×27/2 = 378 potential checks (n). With 4 observations pre-chosen, equation 2 implies the event needed to last 13 days, which in turn results in 702 total checks (C^*), and no necessary self-self checks (S_0=0), again assuming no piers are left vacant. A quick run of **sags.f** on this (28,9,4) combination yielded a schedule with no self-self checks, and no non-zero off-diagonal elements of the check matrix. Could the event have been shorter? Possibly, because if o were set to 3, and not 4, this would yield a duration of only 10 days, no necessary self-self checks (S_0=0) and 405 total checks (C*) and thus the *possibility* that, in those 405 total checks, every single one of the 378 potential checks gets made. However, multiple runs of sags.f using the (28,9,3) combination, even with extremely liberal cost functions, failed to yield a schedule with every potential check made.

We close this section with a note about travel costs. As mentioned, much of proposed practice relies upon gaining efficiencies in light of the expense of an absolute gravimeter comparison. One thing which has not yet been addressed is the overall travel costs of each participant. Specifically it could be argued that one additional optimization constraint could be imposed which seeks to minimize the number of days each meter is actually at the event. As of right now, nothing in any of the GSP has addressed this.

Consider again the ICAG17 example, with 28 meters. For the sake of an example, the authors' chose at random one schedule, as generated from Simulated Annealing which put meter #1 making observations on days 1, 3 and 6 while meter 17 is on days 4, 5 and 10, etc. There is *significant* travel cost for these meters to spend 4 or 6 days at an event when they are only observing on 3 days. In point of fact, the *actual* schedule used in ICAG17 involved "sets" of instruments, that observed over consecutive days (expressly to minimize travel costs): meters 1-9 on days 1-4, meters 10-18 on days 5-8, etc. However, to the authors' knowledge, this schedule was created "by hand," without attempting to rigorously optimize instrument checks as discussed in this paper. Thus one could (in fact, should) devise a GSP which either (a) chooses the cheapest travel costs out of set E_3, or (b) prioritizes cheaper travel costs over "score", and thus defines an optimized schedule as one where the loss of score (that is, a schedule chosen outside of set E_3) comes with an acceptable gain in travel cost savings. Such a problem is left for future work.

15 Appendix D: Summary of variables used in this paper

Below is a quick summary table to assist readers in keeping all of the variables straight throughout this paper. The column "Changes with each schedule" addresses whether, given any particular {*m*,*p*,*o*} set, the variable listed is schedule dependent.

Variable	Description	Integer or Real	Changes with each schedule?	Equation
m	# of meters at the event; also number of diagonal elements in check matrix	Int	No	
p	# of piers used in the event	Int	No	1
o	Minimum # of observations each meter *must* make in the event (while o+1 is the maximum # of observations each meter is *allowed to make* in the event)	Int	No	
d	Minimum number of days in the event for all meters to make at least o observations, no more than o+1 observations, and no piers are ever vacant	Int	No	2
n	# of non-diagonal elements in check matrix	Int	No	7
C	Sum of all elements of check matrix	Int	No	4
S	Sum of all diagonal elements of check matrix	Int	Yes	
C*	Sum of all non-diagonal elements of check matrix	Int	Yes	5
S_0	Minimum possible value of S (= minimum possible number of self-self checks in any schedule)	Int	No	19
C_0*	Sum of all non-diagonal elements of a check matrix when $S=S_0$	Int	No	6
σ	"Score"; Standard deviation of all non-diagonal elements of a check matrix	Real	Yes	
σ_0	"Optimized Score"; Smallest possible value of σ given m,p,o	Real	No	
σ_P	"Perfect Score"; Smallest possible standard deviation of "n" integers which sum up to C_0*	Real	No	18
E	Set of all possible schedules, given m,p,d and disregarding rules about each meter making o or o+1 observations	-	No	
E_1	Set of all possible schedules, given m,p,o	-	No	
E_2	Set of all possible schedules, given m,p,o and with $S=S_0$	-	No	
E_3	"The optimized schedules"; Set of all possible schedules given m,p,o and with $S=S_0$ and $\sigma=\sigma_0$.	-	No	
I_{Lo}	One of the two integers which exist in a set of "n" integers that sum up to C_0* with $\sigma=\sigma_P$	Int	No	9,10
N_{Lo}	The number of times I_{Lo} occurs in a set of "n" integers that sum up to C_0* with $\sigma=\sigma_P$	Int	No	11
I_{Hi}	One of the two integers which exist in a set of "n" integers that sum up to C_0* with $\sigma=\sigma_P$	Int	No	12
N_{Hi}	The number of times I_{Hi} occurs in a set of "n" integers that sum up to C_0* with $\sigma=\sigma_P$	Int	No	13
SPD	Number of possible schedules per day, given m,p	Int	No	25