VCGW

Variable Complexity Grid-Walk

An algorithm for converging on prominent points-of-elevation in multi-modal (multiple peaks and troughs), multi-dimensional (multiple variables affecting the surface's elevation) terrains.

Author:
Leon Zadorin

Date:
August 10, 2012

Centre for the Study of Choice (CenSoC)

University of Technology Sydney (UTS), Australia.

version 0.0.1.1

Contents

1 Background

1.1 Context

For brevity's sake this text shall concentrate on the act of *minimizing* terrain's elevation (e.g. error minimization in multivariate problems). The contrary process of finding the *maximum* – where the convergence on higher points-of-elevation is desired – is semantically equivalent to that of minimization by virtue of terrain's *inversion*. In other words to find a maximum point-of-elevation simply:

- multiply terrain's elevation values by -1 (thereby inverting terrain's surface "inside out" – along the elevation axis);
- find the *minimum* mode of the transformed surface.

1.2 Unimodal terrains

One of the simpler kinds of a terrain is a *unimodal* terrain where – for *all* of the *simultaneously*-considered dimensions – there is only one trough/puddle. In other words, there is no more but one location on a terrain where all of the dimensions' gradients are flat.

For such terrains, even a case as complicated as a multidimensional zig-zagging valley (fig 1, p.3) – down which the "water" would stream should a "light rain" take place – is traversable in a reliable and reasonably computationally-efficient manner by well known algorithms such as BHHH, BFGS et al.

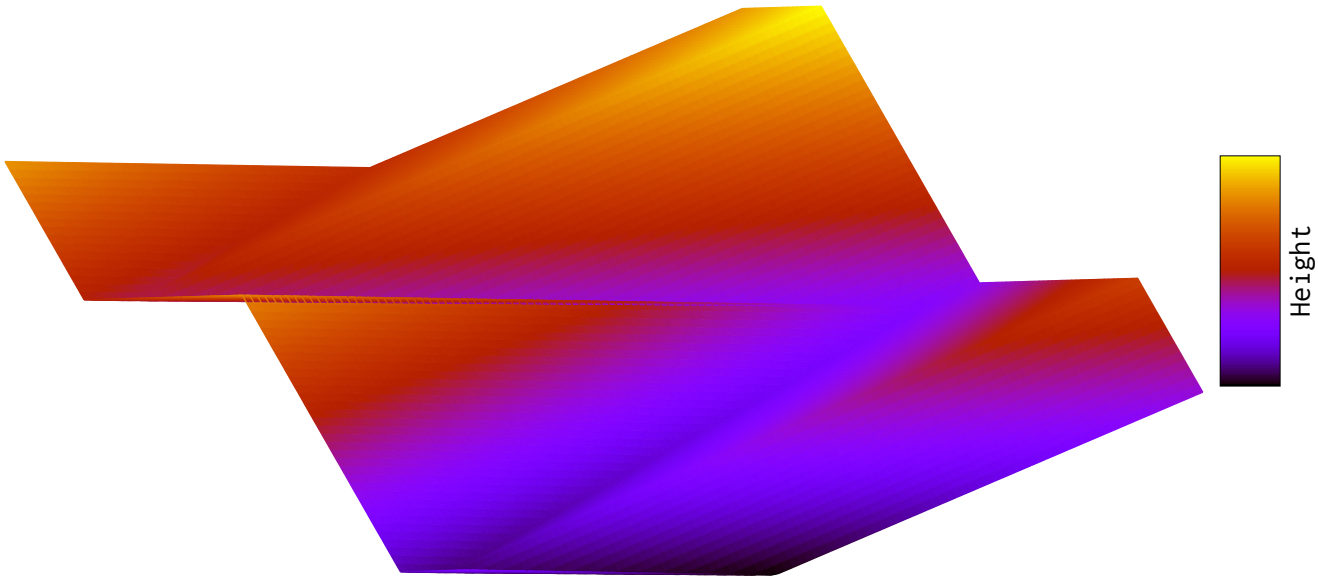


Figure 1: Unimodal terrain of a zig-zagging valley, where rivers flow towards one target.

This is due to the fact that every point on the aforementioned terrain yields a "gravity-suggested" vector $\vec{V}$ whose direction ultimately points towards a lower location (eventuating at the mode in

question). This constitutes the exact reason for the rivers to arrive at one-*and-only-one* mode instead of forming various lakes and puddles.

Such vector-oriented mechanisms also exhibit *unconstrained* convergence semantics. Meaning that no matter where one begins the convergence process – i.e. irrespective of the starting location on the to-be-explored surface – a given algorithm is able to "follow the gravity" and arrive at the lowest point-of-elevation.

1.3 Multi-modal terrains

When one considers a more complicated *multi-modal* terrain – where multiple "puddles" shall form should such a terrain be, once again, subjected to a "light rain" – one cannot rely on the vector-oriented algorithms as heavily, and as robustly, as one would in the case of unimodal terrains. Some of, the multitude of, reasons for the aforementioned statement follow.

1.3.1 No unconstrained semantics

It could be argued that in a multi-modal terrain there simply is no such thing as an "unconstrained convergence".

For instance, if the converging algorithm is exploring terrain's surface in an *apparently* wrong direction – elevation of the surface is increasing as the exploratory sampling-step-size is augmented – then at some point the algorithm must decide to stop following current vector-of-exploration and reconsider it's direction of sampling.

Such a decision, however, could always be *postponed* until the *next* "sample" (e.g. of a greater step-size) – just in case there is a bigger canyon/drop "out there in the distance". Unfortunately such an approach is practically infeasible as it implies for one to *never* stop looking in the current, and apparently-wrong, direction for the fear of missing out on a potentially bigger drop located farther away (fig 2, p.5).

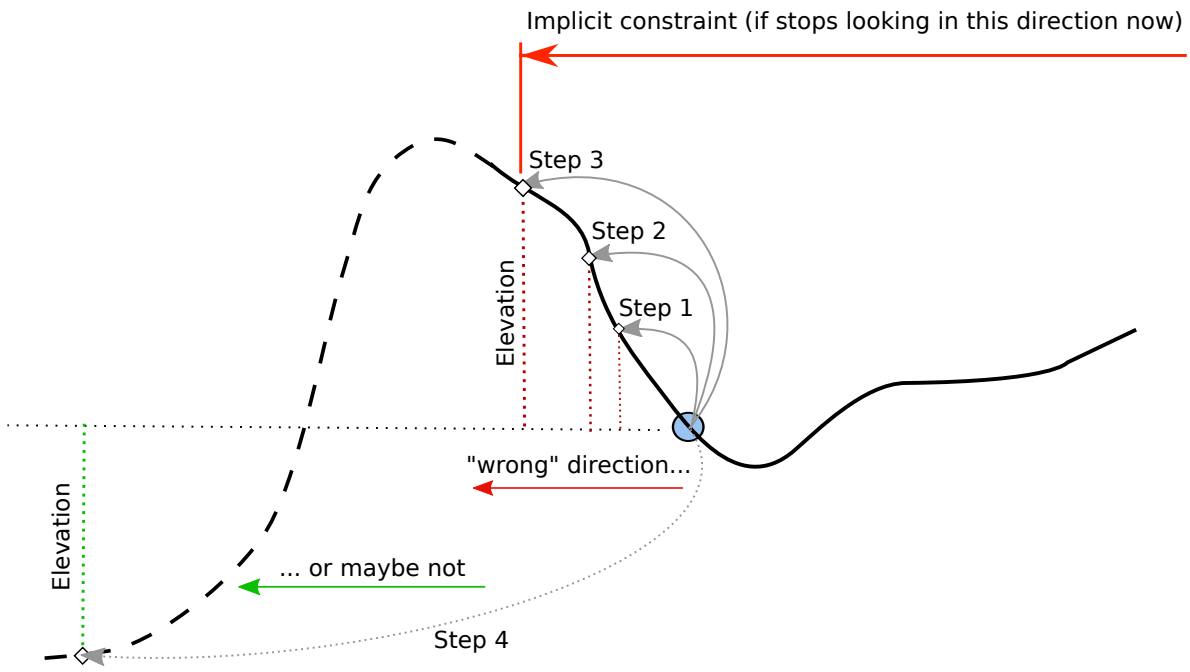


Figure 2: One can never have an unconstrained convergence semantics in a multimodal terrain, for the moment one stops looking in a given direction one is giving up a chance of possibly discovering a better solution.

Moreover, trying to explicitly tune the underlying parameters of vector-oriented algorithms – e.g. initial starting location, step-size and it's reduction/augmentation conditions (such as number of samples taken in the wrong direction and the scale factor by which to change step's span, etc.) – would only translate-to the *implicit* presence of *boundaries*-of-exploration (i.e. the very *constrains* of the convergence process).

This causes one to realize that having an *explicitly*-constrained convergence algorithm in multi-modal terrains is simply more convenient and transparent, if not logical. In other words instead of *inferring* the boundaries from various derivatives such as step-sizes, counts-of-steps before updating direction, etc; one might as well specify the "bounding box" of the *universe-of-interest* which shall subsequently be explored/traversed by the algorithm.

1.3.2 Surrounding sampling points' localization and the misleading consequences thereof

In vector-dependent algorithms, utilization of closely-surrounding sampling points with respect to current location – a traditional approach in establishing the gradient/plane-of-decline – may produce a skew in the exploratory direction because current point's location may be a part of a smaller, other-than-dominant mode.

Conversely – i.e. using a larger distance from the current location – the surrounding points' elevation values may yield an even lesser relevance as *each* of such points may be from a *different* mode thereby forming incorrect overall angle of the gradient.

Consequently – due to the acuteness of the error in the direction's angle/diagonality and it's balance with respect to dominant mode's width/broadness – the aforementioned issues may cause the

algorithm to "swoosh-by" the optimal terrain coordinates, or be trapped in the suboptimal location.

Moreover, the inefficiency of such an approach is further compounded by the fluctuating nature of step-size expansion/shrinkage – doubling or halving the step-size may prove to be catastrophically incorrect depending on the current position on a traversal graph.

1.4 Conclusion and possible improvements

As a consequence of the aforementioned thoughts one may arguably conclude that the very benefits of vector-oriented algorithms, whilst being so self-evidently present in unimodal terrains, are more of a liability rather than an asset in multi-modal scenarios.

The various existing strategies (e.g. simulated annealing) designed to "jump out" from the local minima (as a remedy for the aforementioned issues) could also be seen as, somewhat, suboptimal.

Firstly, lesser dependence on the direction/vector-oriented exploration in such algorithms usually comes at a price of being heavily leveraged in random distributions. Randomness, however, is extremely exhaustive in terms of computational requirements – to the point of becoming either practically infeasible, or yielding insufficient efficacy, when applied to highly multidimensional terrains.

Secondly, the act of "jumping-out" could be thought of as a process of *zooming-out* (with respect to the exploratory confines of an algorithm) from within a *previously* zoomed-in (local) minimum. Following such a conceptualization it is possible to consider the act of zooming-out as an indication of a *premature* zoom-in in the first place. In other words, simply performing a more comprehensive exploration at a *current* zoom-level (*before* zooming-in) may yield a greater efficacy with respect to speed and robustness of the convergence processes. The act of zooming-in, then, becomes a unidirectional process – no back-and-forth, "washing-machine" cycles of zooming-{in, out}.

1.4.1 Cornerstone proposition – not as complex as pure randomness

The notion of a more comprehensive exploration at a given zoom-level extrapolates quite intuitively towards the need for a more efficient design when specifying a set of sampling locations – a set which converging algorithm shall explore – in order to minimize the number of sampling steps whilst retaining the desired robustness in finding a better minimum.

The distribution-based selection process from a discrete search space – a space usually representing a *complete* pool of *all* of the possible permutations across *all* of the dimensions – for the to-be-evaluated sampling location is inefficient with respect to a *highly*-multidimensional terrain.

Naturally such a statement is implicit for the purely random terrains (e.g. a *30*-dimensional noise). However, the aforementioned inefficiency of such a selection process is also applicable to "not as complex as pure-randomness" landscapes.

For instance, consider a valley of an acutely 4-dimensional diagonality (zig-zagging characteristics) positioned in a *30-dimensional* space – a valley, that is, whose direction-of-decline is highly varied along 4 dimensions whilst remaining reasonably stable and not abruptly-changing with respect to the remaining dimensions at a given point; which dimensions are highly volatile and which are stable is not explicitly known prior to the deployment of the convergence process – something, in laymen terms, that is not noise but at the same time is complex-enough to yield a problem for the vector-based convergence algorithms.

Given an astronomically large size of a *complete* set of coordinates (i.e. all possible locations in a *30*-dimensional space), the comprehensiveness of even an extreme multitude of distribution-based draws from such a set is likely to yield a low probability of finding anything particularly meaningful when traversing the valley. In other words, the found solution is likely to diverge away from an acceptably good one – let alone the best possible one, in the confines of a given bounding-box/constrains-set – rather dramatically.

Of various improvements to the issues raised in previous paragraphs, one is to leverage the distinction between the n-dimensional complexity of the valley (e.g. $n = 4$) and the overall N-dimensional makeup of the whole space (e.g. $N = 30$) within which the valley-in-question resides – thereby appropriating/optimizing the generalized convergence process to a terrain which, whilst still being quite multi-modal and complex, is distinguished from that of a random noise.

In other words instead of looking at a set of all possible coordinates in a *30*-dimensional space at once, the improved algorithm needs only to consider a set of all possible coordinates with up-to any *4* dimensions being modulatable at once (away from the current location/center-of-exploration). In such a scenario, the combinatorial modulation of "n-dimensions-at-once" necessarily acts as a predicate for traversing an acutely n-dimensional zig-zagging/diagonality of the valley without getting caught in various suboptimal modes/troughs (the exact and illustrative description of how the question of diagonality relates to the "n-at-once" coefficients modulation shall be addressed later on in this text).

Moreover, reducing the number of the to-be-explored locations implies a more succinct selection of which points to include and which to discard, eventually deprecating the need for random draws – every one from the optimized set of to-be-explored points needs to be evaluated.

Generally speaking, the aforementioned process (as shall be elaborated upon in greater detail later on in this text) yields a grid-divided problem-space with a pre-determined set of to-be-explored complexities (e.g. combinations of all possible grid coordinates of up to any 4 modulated dimensions at once in a 30-dimensional universe).

The overriding notion being that all of the possible combinations of n-modulated-coefficients and their values in the N-dimensional space (e.g. $n < N$) produce lesser number of evaluations when compared to cases where *all* of the N coefficients are modulated at once. This of course is not always the case and there are times when, with sufficient computational power, one is able to prefer complete, N-modulated-coefficients at once, "brute-force" approach. Table 1, p.8, attempts to illustrate but one such a case.

n-dimensional complexity	number of evaluations	deficit/surplus in relation to 2^{30}
1	60	1073741764
2	1740	1073740084
3	32480	1073709344
4	438480	1073303344
5	4560192	1069181632
6	38001600	1035740224
7	260582400	813159424
8	1498348800	-424606976

Table 1: Number of evaluations needed in the context of a 30-dimensional universe with 2 grid-points -steps per dimension.

It must be stated, however, that the aforementioned example is rather unrealistic with respect to practically applicable settings. Namely, using only 2 grid-points per dimension will, in a vast majority of cases, yield insufficient precision with respect to identifying smaller fluctuations/widths in the underlying terrain (the reasoning for such a statement will become implicitly clear as this text progresses further towards the description of the finer points of the algorithm et al.). Extrapolating a 30-dimensional problem space to just the 3 grid-points per dimension yields a different picture as per table 2, p.8.

n-dimensional complexity	number of evaluations	surplus in relation to 3^{30}
1	90	205891132094559
2	3915	205891132090734
3	109620	205891131985029
4	2219805	205891129874844
5	34628958	205891097465691
6	432861975	205890699232674
7	4452294600	205886679800049
8	38401040925	205852731053724

Table 2: Number of evaluations needed in the context of a 30-dimensional universe with 3 grid-points -steps per dimension.

Whilst anticipating the exact level of complexity of a given terrain is not guaranteed priori the start of the convergence process – i.e. is it an acute diagonality of up to four-dimensions, or is it five dimensions or even more, etc. – it is possible, as once again shall be elaborated upon in greater detail later on, to test posteriori the convergence run as to whether the supposed terrain's complexity has been sufficiently traversed/explored by the current settings of the converging algorithm; and, if necessary, "dial up" the algorithm's complexity to that of a greater multi-dimensional robustness and then repeat the whole of the convergence process.

2 The algorithm – VCGW

2.1 Introduction

The algorithm presented in this text (VCGW) is designed to exhibit a lesser dependence (if at all) on the issues described so far. Namely, the difficulties in calculating the exploratory vector (via elevation values of the closely-surrounding points); or the management of the sampling step-size along the established direction.

Overall, the algorithm in question:

- does *not* deploy a vector-based convergence process;
- is an *explicitly constrained* exploration algorithm – boundaries are stated *explicitly* as part of the parametrization and tuning of a given convergence run.

More specifically, the terrain's exploration area is divided into a grid of evenly-spaced sampling points. The sampling points – yielding discrete elevation values on the continuous surface of the terrain – are subsequently evaluated by VCGW in order to determine further localization and zoom-in/contraction of the boundary-constraints of the exploration area.

Such a sampling approach stands semantically closer to the concept of a sampling-resolution or sampling-rate – not too dissimilar from the aspects of Nyquist theory in the domain of DSP (Discrete Signal Processing) – and departs from the notion of having to perform calculations based on the epsilon (smallest-feasible) distance from the current point in order to correctly depict the shape of the underlying terrain.

In other words, the algorithm's capability to detect finer fluctuations in terrain's elevation (e.g. narrow/sharp troughs) is determined by how fine the sampling grid happens to be – much as the DSP's capability to faithfully depict a band-limited signal is based on the frequency (period/step-size) of the sampling rate.

Of course – given the grid-based nature of the terrain's exploration/sampling – one may suggest that such an approach (even when applied in light of the aforementioned cornerstone proposition) is *still* extremely slow in terms of numeric efficiency as it shall yield an invariably greater number-of-evaluations as compared to more traditional, vector-based, algorithms. After all, there is no gradient-based priority given to various sampling points and no "hopping over" greater distances via sample's step-size modulation – as may well be the case with the BHHH et al.

Before proceeding any further, however, one ought to point out the lateral implications of the aforementioned statement. Namely that a comparative slowdown in computational-speed implies a greater robustness-in/quality-of convergence.

In other words, a more primitive grid-like sampling mechanism of exploration yields a greater robustness in terms of multi-modal terrain minimization (due to its lesser dependence on the issues described earlier in this text); and so, arguably, it is not the slowdown of a grid-based convergence but rather it is the speedup at a price of a lesser robustness – as exhibited by the vector-based algorithms – that ought to be the point of contention and in need of being re-considered in the context of multi-modal terrain exploration.

To put it another way — it would be incorrect to accelerate the speed by compromising on the quality. Instead, one takes a step back in speed to further the gains in robustness. The speed is recouped

later, at a different (implementation) stage of the algorithm's life-cycle. Namely, being capable of evaluating different sampling points *at the same time (simultaneously)* allows VCGW to leverage a *heavily-parallel computational environment* which, in turn, upscales computational efficiency of the overall convergence process in virtually unrestricted manner.

For instance, in a small-to-medium office there may be 35 computers, each with a quad-core CPU, yielding a capacity to evaluate some 140 sampling points *simultaneously*. Expand such a scenario to a collaboration with another organization/office – either locally or via the Internet – and what eventuates is a *massively parallel* convergence run with a greater robustness due to grid-like exploration and an improved speed rebate due to computational parallelization.

2.2 Walking via coefficients' modulation

At this stage, let us depart from the above digression and return to describing VCGW in greater detail.

The algorithm, essentially, relies only on any one sampling location (in all of the multidimensional space) exhibiting lower-terrain characteristics (as compared to the current point/location of reference). In other words, no Hessian matrix or gradients are needed. As has been already stated – it is a vector-*independent* process and, more succinctly, it is a *grid-dependent*, *walking* algorithm. The behavior of the algorithm is to sample the aforementioned grid-points until one with a lower-than-current-point elevation is found. At such an instance, VCGW walks/re-centers to the newly-found point and repeats the process.

As a matter of further computational efficiency – within the aforementioned context of n-dimensional diagonal complexity of the terrain's surface and the overall, N-dimensional, nature of the model (where $n <= N$) – the algorithm has an option to chose it's next sampling point by modifying *fewer than n-coefficients-at-once* with respect to the current location on the terrain.

For example, given a *30*-dimensional model with a *10*-dimensional diagonal complexity, VCGW may optionally start by considering the sampling locations with only *one* modified-coefficient with respect to the algorithm's current location, i.e. start with only the *one-coefficient-at-a-time* modulation:

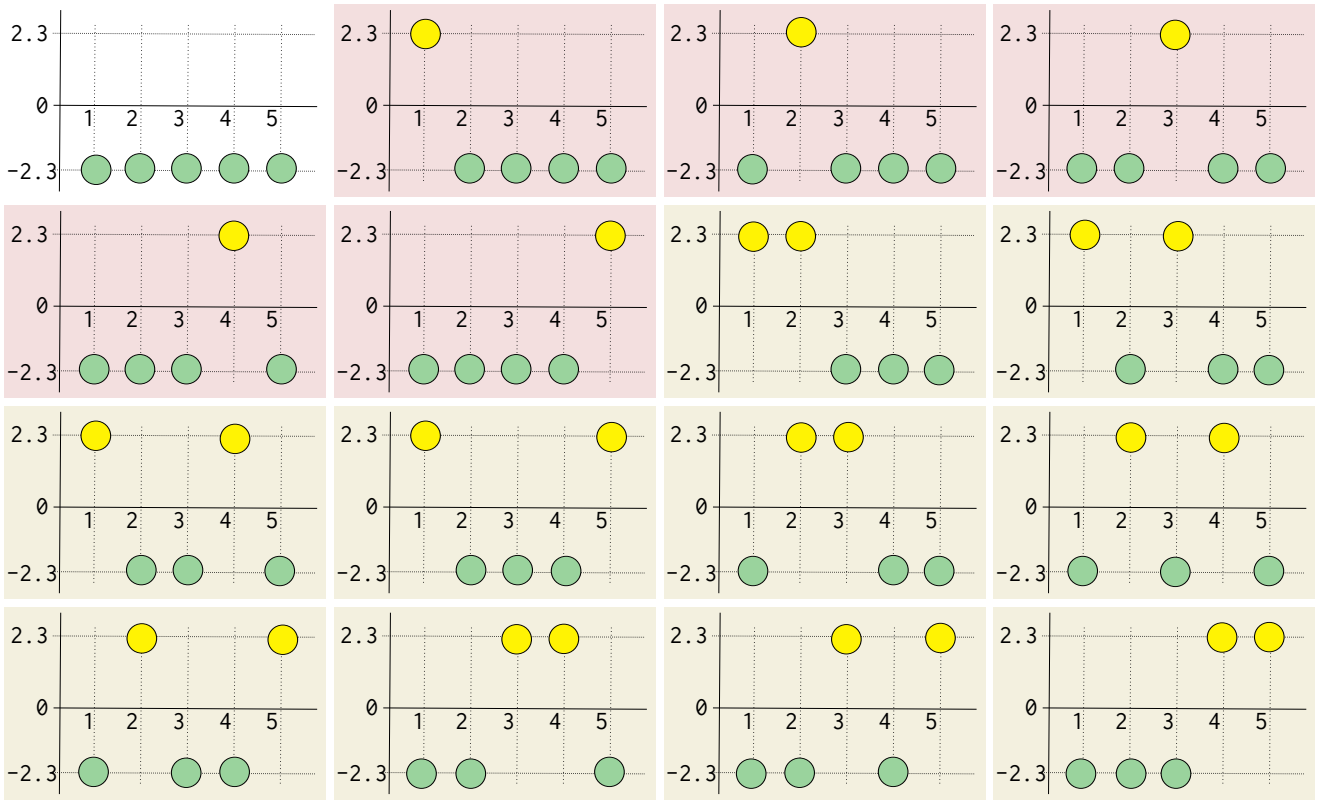
- take the 1^{st} coefficient and move it to the next point on the grid, away from current location; then evaluate the resulting location for it's elevation value;
- if no better elevation was established, repeat for the next point on the grid of the 1^{st} coefficient thereby exhausting all of the grid-locations for the coefficient in question;
- if no better elevation was established, *revert* the 1^{st} coefficient to it's original value (i.e. back to current location/point-of-reference) and repeat the whole process with the 2^{nd} coefficient being modified in a similar fashion;
- and so on, and so on...

Basically, the algorithm exhausts all of the possible combinatorial complexities (i.e. all of the possible locations) where any one (but *only* one at a time) coefficient has been modified with respect to the current coordinates.

When *2-at-once* coefficients are being modified (for the to-be-evaluated sampling point) the algorithm considers and explores all of the combinations of such modifications. For example, the following collection of combinatorial representations would be exhausted:

- firstly all of the possible combinations where coefficients {1, 2} are modified along their respective grid-points are evaluated (for better elevation value than "at a current location"). For example, if both coefficients had three grid-points dividing their boundary-constraints, then there would be $3^2 = 9$ total combinations to be evaluated;
- then, if no better elevation was found, coefficients {1, 2} are returned to their original values and coefficients {1, *3*} are modified and evaluated;
- the process proceeds, in a similar fashion, to evaluating pairs of coefficients such as {1, *4*}, {1, *5*} and so on until {1, *30*};
- next, once again if no better elevation is obtained thus far, combinations such as: {*2*, 3}, {*2*, 4}, {*2*, 5} and so on until {*2*, 30} are processed;
- intuitively, the next set of combinations to consider is: {*3*, 4}, {*3*, 5}, {*3*, 6} and so on until {*3*, 30};
- progressing in such a fashion, the process reaches sets of combinations such as: {28, 29}, {28, 30};
- and eventuates at the final pair of {29, 30}.

Visualized example of *one-at-a-time* and *two-at-a-time* combinatorial modifications – in a somewhat simplified model, made up from 5 coefficients/dimesions and only 2 grid-points per coefficient – is illustrated in fig 3, p.12.



Coefficients/dimensions are shown along the horizontal axis and the coefficients' grid-based values are along the vertical axis. Green color indicates original, unchanged (current location's) coefficient; yellow illustrates a modified coefficient. For brevity's sake, some simplifications are assumed:

- there are only 2 grid-steps per coefficient/dimension and their values are identical for all of the dimensions. In reality, of course, there will be a greater variety/heterogeneity between how many grid-steps are in a given dimension, which values/range the dimension's grid-steps occupy, etc.;
- the current/starting point for all of the coefficients' values is the first/lowest grid-value. In reality this will not be the case as the starting location is selected at random for each dimension independently – i.e. which starting grid-point to use for a given coefficient at every zoom-in stage is chosen at random – this, however, is related to different issues discussed elsewhere in this text.

Figure 3: Visual conceptualization of *one-at-a-time* followed by *two-at-a-time* combinatorial modifications.

If no better find is obtained in a given *x-coefficients-at-once* combinatorial context, VCGW increases the number of simultaneously-modified coefficients and then explore all of the possible permutations (sampling points/steps) in such a new context. For instance, if *one-coefficient-at-once* context fails to yield a better find, the algorithm may progress to *2-at-once* combinatorial modulations, and then to *3-at-once* and so on – until an explorable limit of *n-at-once* (e.g. $n = 10$) is reached. Essentially then $1 <= x <= n$.

So far all of the aforementioned illustrations and examples had covered instances without a better elevation being obtained whilst evaluating various sampling points. If, on the other hand, a better find is obtained (at any stage of the aforementioned sampling process) then the algorithm *re-centers/walks* to the newly-found location and resets the calculation/exhaustion of the combinatorial representation sets. An example of such a resetting scenario follows:

- the process starts with a *one-at-a-time* combinatorial context;
- the algorithm subsequently proceeds to *2-at-once* and then to *3-at-once* context (due to a lack of better elevation obtained thus far);
- then, during *3-at-once* exploratory context, a better elevation value is found;
- the algorithm re-centers to the newly found location;
- the algorithm *also* resets its combinatorial context back to it's initial/starting complexity-level (which in this case happens to be a modulation of *one-coefficient-at-a-time*);
- the whole process is then repeated (but this time from the new/walked-to location) – e.g. *one-at-a-time* coefficient modulations are made; if no better find is obtained then the process escalates the combinatorial complexity to the *2-at-once* context; and so on.

2.3 Zoom-in and finalization

Eventually, VCGW exhausts *all* of the to-be-explored sampling points – combinations of grid coordinates of up to and including the chosen practically-feasible limit of n-modified-coefficients-at-once in a N-dimensional model (where $n <= N$) have all been evaluated.

What follows is the act of *zooming-in* – the boundaries around the current location (which exhibits best elevation thus far) are shrunk and the whole of the aforementioned exploratory process (but on a smaller scale due to shrunk exploratory space) is repeated – i.e. the whole of the convergence run is done as if nothing had preceded it:

- newly-shrunk, boundary-defined exploration area is divided into a grid of sampling points along every dimension;
- $\{1, 2, \ldots, n\}$-coefficients-at-once explorations are performed as VCGW looks for, and walks-to, locations with the lowest elevation thus far;
- if no better find is obtained (after all of the combinatorial n-at-once representations in a N-dimensional model have been evaluated) the algorithm zooms-in again.

The finalization (termination) criteria for the overall convergence process is somewhat similar to the traditional algorithms. Namely, if the current "zoom magnification" – i.e. the size of the current exploratory-area as designated by the explicitly-constrained -shrunk boundaries – is small-enough not to make a feasible difference to the estimation process (insofar that changes in the newly-found location will yield an irrelevantly-small amount of difference with respect to the *already-found coordinates*) then the convergence process is deemed to be complete.

As a matter of algorithm's flexibility in configuration and, indeed, common sense – the number of grid-points dividing various dimensions is independently configurable for each of the coefficients (e.g. there may be 5 grid-points per first coefficient and only 2 grid-points per second coefficient). Moreover, each zoom-level and *each combinatorial complexity* (e.g. x-coefficients-at-once) is able to have its own *individually-adjusted* number of grid-points per any of the dimensions.

2.4 Miscellaneous characterization

2.4.1 "Sampled points" memory

To further the traits of computational efficiency, as VCGW goes about its business of evaluating various sampling locations it also *retains the memory of each and every one of the previously evaluated sampling locations*. This stands true even for the cases which did not exhibit a better elevation with respect to the then-current central point of reference (i.e. even if the algorithm did not "walk" to the point in question). Basically, like a diligent tourist, VCGW keeps the record of every evaluated location in the model's universe.

The coordinates of every "sampled place" are stored not with respect to the current location (i.e. which coefficients have their values modified away from their "current/default" values at a given moment), but rather in an *absolute* – unique to the complete multidimensional context of the model – form (i.e. every coefficient's explicit value is remembered). This allows VCGW to determine if the about-to-be-sampled point with respect to the current location has already been evaluated (even if as a matter of previous steps from *any of the older* locations of reference).

In other words, as further sampling occurs from the currently walked-to point – the future candidate sampling points are only those that do not represent any of the previously sampled locations (even if such were evaluated from a different reference location).

2.4.2 Randomly selecting starting locations

As alluded to earlier, the starting location (from which the sampling process begins it's exploration) is chosen at random. This, in fact, is done at the beginning of every zoom-in stage so as to facilitate a greater certainty in determining whether the finally converged-on location represents the global mode (as opposed to being a local minimum).

The rationale behind such an approach is that in a heavily multidimensional context (e.g. 30 dimensions) even a small number of grid points used to partition model's exploratory space (e.g. only 2 sample points per each dimension) shall result in a *vast number of different starting locations* (e.g. $2^{30} = 1073741824 \approx 1$ *billion*). Combined with the idea of performing *multiple complete convergence runs*, the aforementioned high entropy of starting locations leads to a proposition which states that *if* various convergence processes (each started at random from a vast pool of different locations) *all lead to the same, final, converged-on point* – then in all likelihood such a converged-on point *is* the dominant mode.

Otherwise, had the finally discovered location not been the dominant one, then the multiple convergence runs would *most likely* produce *differing* converged-on locations. In other words, given a billion different starting locations at every zoom level, if the final converged-on mode was *not* a dominant one than it would not be identified as such by *multiple, randomly-started* convergence runs.

In fact, here, the heavily-multidimensional nature and overall complexity of the model act as a positive reinforcement for the confidence in the final solution. For instance, if the number of possible starting locations grows from 1 billion to 100 billion – either due to additional dimensions being introduced to the model or due to a finer grid dividing the model's space – the more *unlikely* it would be for the various convergence runs to yield the same location if it isn't representative of the dominant mode; and therefore the more confident one is in the final solution being the most optimal one if various convergence runs do end up in the same location.

The overall process could essentially be outlined as follows:

- Perform a *complete convergence* run (i.e. all the way up to the termination criteria). In other words, go from the outmost zoom-level (as depicted by the starting constrains of the algorithm's controlling parameters) and keep zooming-in (randomly-selecting starting locations at each of the zoom levels) until the algorithm reaches it's termination criteria and the solution is deemed as being "converged-upon".
- Repeat the whole process again (i.e. a complete convergence run) thereby increasing the quality-assurance in the resulting converged solution.
- Keep repeating complete convergence runs until eventually obtaining satisfaction regarding the overwhelming certainty in the final solution.

In other words, one is able to empirically test for the robustness/success of the convergence process with the help of randomly-chosen starting locations and multiple convergence runs.

Naturally, multiple convergence runs may not yield the same converged-on location. This could mean that the configuration parameters need to be adjusted (e.g. the range of boundaries/constraints; n-coefficients-at-once combinatorial complexity; number of sampling points per dimension; and so on). As was stated earlier in this text, anticipating the exact level of complexity of a given terrain is not guaranteed priori the start of the convergence process and so it is not possible to predict the exact depth/complexity for the controlling parameters of VCGW.

However, what the aforementioned process (i.e. repeated convergence runs with randomly chosen starting locations) does is enable one to *test posteriori* the convergence runs as to whether the supposed terrain's complexity has been sufficiently accommodated by the algorithm's current settings. If it becomes apparent that the current settings are not sufficiently-robust to match the anticipated complexity of the underlying terrain then one is free to "dial-up" the algorithm's controlling parameters to a greater depth at the expense of longer computational times.

It could, of course, be that a model is simply too complex for the given parallel-computational environment (the act of minimizing a random-noise terrain over 30 dimensions; or cracking passwords of high encryption-complexity). VCGW is by no means a universal "silver bullet" for any imaginable kind of a problem. Rather, it is simply intended to be of a more optimal solution for the specific/limited kinds of problems (e.g. aforementioned n-dimensional complexity in N-dimensional context where if N is a large number then $n < N$).

As a final comment regarding the confidence in the final converged-on solution, one may point out that the total dimensionality of a model may not be particularly high (e.g. a model may only have 4 dimensions) yielding a somewhat smaller pool of starting locations and thereby leading to a lower confidence in the aforementioned *probability*-based quality-assurance. In other words, the likelihood of obtaining *differing* solutions from the terrains without a dominant mode wouldn't be anywhere near as high had the same process been applied to a 30-dimensional model.

In such a case one can still derive confidence in the final solution – not necessarily from the multitude of convergence runs but rather from the knowledge that the highest x-coefficients-at-once combinatorial complexity can easily approach the *total* complexity of the model (e.g. 4-coefficients-at-once would completely exhaust all of the possible grid-locations in a 4-dimensional model).

2.4.3 Multiple-coefficients-at-once starting contexts

Given the algorithm's capability to be deployed in massively-parallel computational environments, the beginning of the convergence process with *one*-coefficient-a-time combinatorial complexity is *optional* and the convergence process may well start it's run with a *multiple*-coefficients-at-once complexity.

For instance, given the following characteristics of a hypothetical case:

- exploratory process is limited to a maximum of *4*-coefficients-at-once combinatorial complexity;
- 5 grid-points are used per each of the dimensions;
- 170 equally-powerful computational nodes (e.g. computers or computing CPU cores) are used to perform calculations

one is able to observe that

- the maximum number of sampling points at *4*-coefficients-at-once combinatorial complexity is $5^4 = 625$
- the total *3*-coefficients-at-once combinatorial complexity yields $5^3 \times 4 = 125 \times 4 = 500$ sampling points, due to the following:
  - a modulation of any given *3*-coefficients-at-once combination yields $5^3 = 125$ combinations
  - and there are in total 4 possible combinations of any 3-coefficient groupings from the total of four coefficients: {1,2,3}{1,2,4}{1,3,4}{2,3,4}
- the total *2*-coefficients-at-once combinatorial complexity yields $5^2 \times 6 = 25 \times 6 = 150$ sampling points, due to the following:
  - a modulation of any given *2*-coefficients-at-once combination yields $5^2 = 25$ combinations
  - and there are in total 6 possible combinations of any 2-coefficient groupings from the total of four coefficients: {1,2}{1,3}{1,4}{2,3}{2,4}{3,4}
- the total *1*-coefficient-at-once combinatorial complexity yields $5^1 \times 4 = 20$ sampling points

Given a presence of 170 computational nodes, one may well chose to start the exploratory process with *2*-coefficients-at-once combinatorial complexity so as to minimize the "doubling-up" of identical computations being performed by multiple computers. Otherwise, processing *1*-coefficient-at-a-time (i.e. 20 sampling points) with 170 computers will yield possible underutilization of 150 computational units.

In other words, one is able to generalize the starting combinatorial complexity of the VCGW as an x-coefficients-at-once scenario, where $1 <= x <= n$.

In reality, the aforementioned arithmetic may not be as clear (e.g. different computers may vary drastically in their number-crunching power or be used in diverse scenarios where the CPU power is being shared with other tasks outside the algorithm's purpose) but the thesis of being able to start the convergence process with multiple-coefficients-at-once modulation still stands to reason.

2.5 Further rationale behind VCGW

At this stage it is possible to elaborate a little further on the finer points of the algorithm's characteristics, namely:

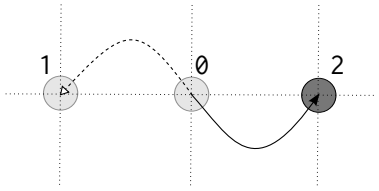
- why there are options for escalating the *x-at-once* modified-coefficients contexts (from the standpoint of efficiency in the path trajectory/exploration). In other words, why x is increased in value up to the limit n (i.e. the limit of the practically feasible combinatorial complexities) – as opposed to simply performing one set of explorations of l-at-once complexity *right from the start* (i.e. instead of going through process of $1 <= x <= n$, why not simply set $x = n$ and thereby evaluate only the n-at-once combinatorial complexity where n coefficients are participating in the makeup of the complete set of the sampling locations);
- why recenter the point-of-reference immediately (on the *first* better-elevation find) – as opposed to exhausting all of the sampling points in a given x-at-once context and only then re-centering to the lowest-of-all-found locations;
- why grid-based spacing of the sampling points is preferred over a random-based spreading of sampling locations. In other words, why not just chose a sampling point along a given dimension via a random draw from the boundary-constrained range as opposed to dividing such a dimension in an evenly-spaced grid and only then considering such "grid-snapped" locations for their elevation values.

2.5.1 Escalation of *x*-at-once modified coefficients

Compare *worst*-case scenarios for the path trajectory – the highest number of samples needed to discover a point of lower elevation – apropos starting with *one*-coefficient-a-time vs starting with *all*-coefficients-at-once complexities (in this context, all-coefficients-at-once implies *starting* the exploration with *all* coefficients' values being considered *at once* when creating an index of all possible sampling locations).

Naturally, such a comparison reduces to a moot point when considering 1-dimensional models – i.e. where terrain's elevation effectively changes along one dimension only. In such a case, all-at-once *is* one-at-a-time and both approaches yield a maximum of 2-steps traversal path. Fig 4 p.18 attempts to illustrate such a point – even if only to establish a baseline for further arguments – the utility of which shall improve as the discussion progresses further to consider multi-dimensional models.

- Bird's eye-view of the terrain (lighter shade indicates higher elevation, darker color implies lower elevation):



- Perspective representation:

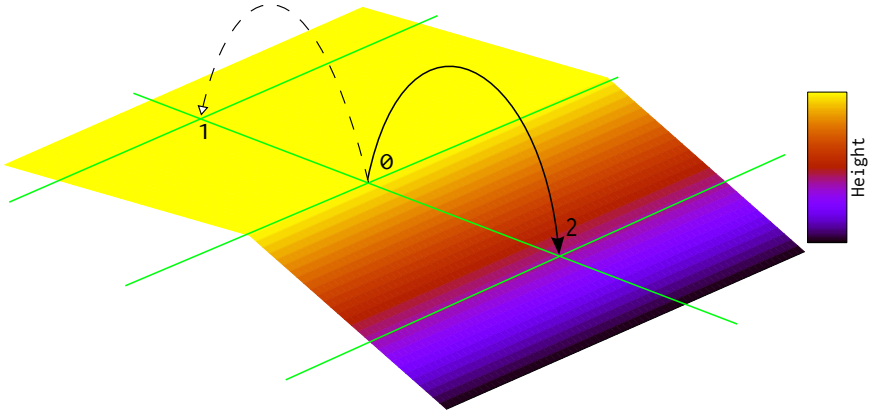
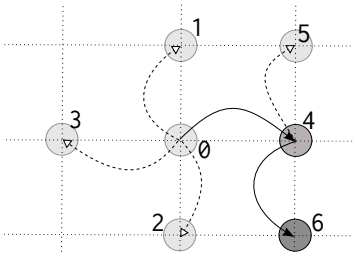


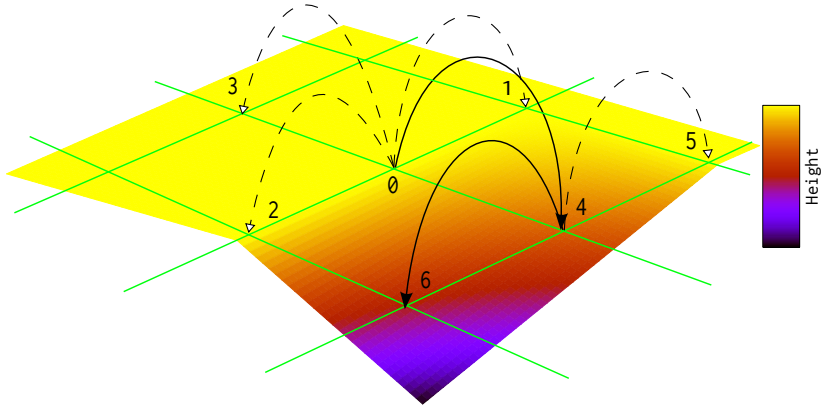
Figure 4: 1-dimensional trajectory path for highest number of steps needed to find a better mode

Following-on from the above observation, one is able to apply the comparison between one-at-a-time and all-at-once approaches to a *two*-dimensional model. In such a case, the difference between two approaches becomes more obvious with one-at-a-time approach yielding the maximum of 6 steps whilst all-at-once approach requiring the maximum of 8 steps (fig 5 p.19).

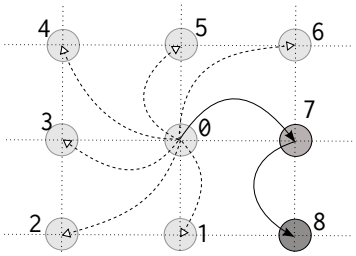
- One-at-a-time approach in 2-dimensional model: maximum of 6 steps
  - Bird's eye-view:



  - Perspective representation:



- Starting convergence with all-at-once approach in 2-dimensional model: maximum of 8 steps (both coefficients are considered when making up a possible index of exlporatory locations)
  - Bird's eye-view:



  - Perspective representation:

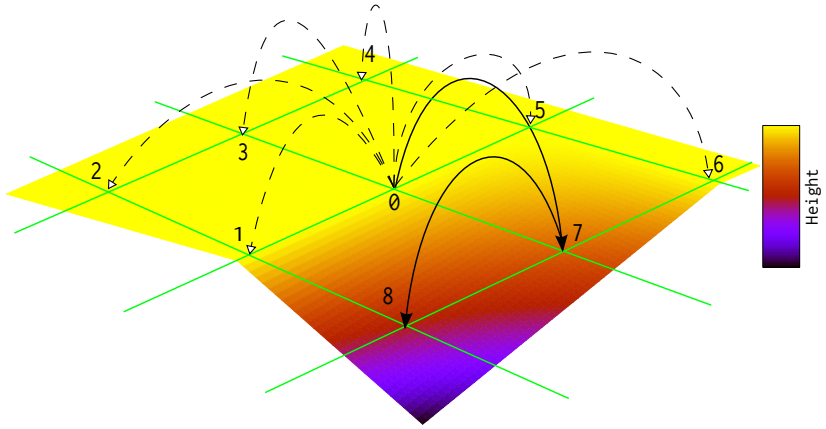


Figure 5: Two-dimensional comparison between *1*-at-a-time and *2*-at-once approaches

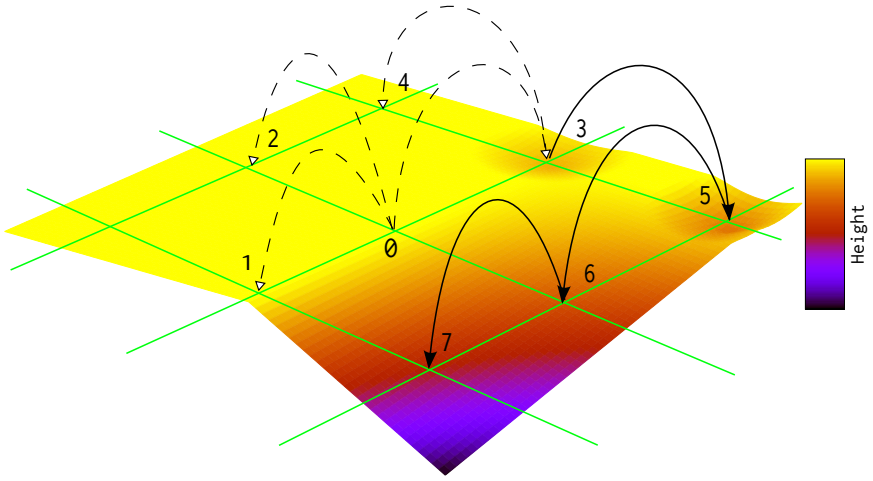
When such a concept was augmented, by conducting experimental convergence runs with greater number of coefficients (e.g. models with 16 dimensions), the empirically-observed behavior appeared to further emphasize the differences in computational efficiency between *one*-at-a-time and *all*-at-once modulation contexts. In other words, whilst modifying all-coefficients-at-once, the system (i.e. an implementation of VCGW in a form of a computational software) was taking much longer to find a better location, as opposed to the time taken to obtain a better find with one-coefficient-at-a-time approach (not a particularly unreasonable observation considering that, even with only two grid-steps, a 30 dimensional model would yield approximately one billion possible sampling locations with all-at-once combinatorial complexity, escalating to approximately 931,322,575 trillion sampling locations if 5 grid-steps were used per each of the dimension).

Of course, turning one's attention to a *best*-case scenario with respect to the trajectory path – i.e. the least number of steps VCGW may take in order to arrive at a better point of elevation – one is able to note that all-at-once approach yields a better mode in just the one step. Nevertheless, as number of coefficients grow (e.g. to 30) the probability of "one-step" scenario taking place becomes microscopically small (e.g. one in a billion). Indeed, such a notion has been empirically accounted-for – the act of modulating all-coefficient-at-once in a 30-dimensional model had resulted in practically-infeasible periods of time where the algorithm would eventually "hang" whilst attempting to find a better mode of elevation due to an enormous pool of points in need of sampling and a very small chance of actually stumbling on a *best-case-scenario* candidate.

2.5.2 Recenter the point-of-reference immediately on the first found improvement

On the subject of why recenter immediately, as opposed to fully exhausting all of the sampling locations (in a given x-coefficient-at-once context) and only then making a move, it could be argued – as illustrated by fig 6 p.21 – that immediate re-centering is indeed suboptimal and, in fact, exhausting all of the sampling points in a given x-at-once context before any re-centering to the lowest-of-all found location yields a trajectory path with *lesser* number of steps.

- Immediate re-centering, yielding 7-steps worst-case trajectory path (with respect to the terrain's shape – presume elevation at following step numbers to be in descending order: 3, 5, 6, 7)



- Exhausting all of the current complexity's locations *before* re-centering yields a 6-step trajectory path (with respect to the terrain's shape – presume elevation at following step numbers to be in descending order: 1, 5, 4, 6)

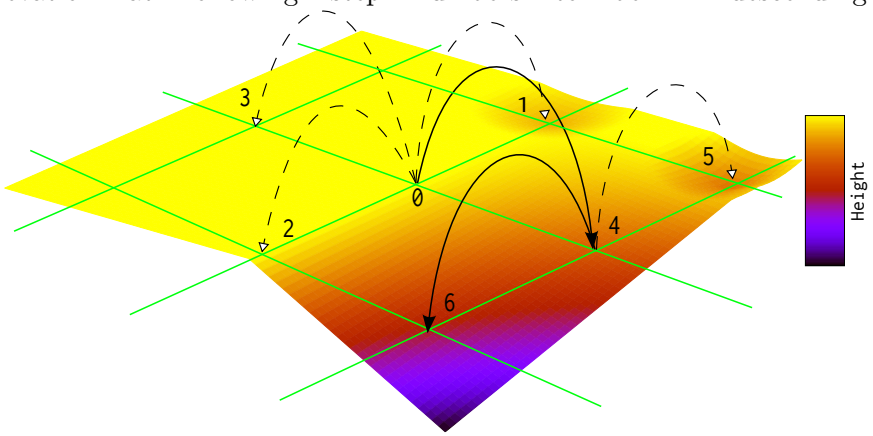


Figure 6: Immediate re-centering vs exhausting all of the locations (in a given compexlity) before re-centering

Having said this, there are a number of caveats to the aforementioned observation:

- Exhausting all of the available sampling steps from a given central point may appear simple-enough in *one*- or *two*-coefficients-at-once contexts, however, the complexity of such an exhaustion shall rise dramatically with an increase in the number of simultaneously modified coefficients. For instance, a 16-coefficients-at-once context – even with as few as 3 grid-steps per dimension – shall yield over 43 million evaluations *before* re-centering (i.e. before making a step in a grid-walking convergence process). Consequently, the act of re-centering as soon as a better find is established is *also* implicitly an act of *resetting* the x-coefficients-at-once complexity where the majority of exploration is done with *fewer* simultaneously-modulated coefficients, and the computationally-taxing increase in the number of coefficients-at-once complexities is done on *rarer* basis – i.e. when no other option is available at a lower x-coeffs-at-once complexity.

- Ultimately, and of a considerably greater importance, it is really a question about *diagonality*. A given starting point – as shown in fig 7 p.22 – could be well off the grid's center, which is rather obvious. This would not only yield a potentially greater number of samples/hops to get to the dominant mode; but also a *complete entrapment in a local mode* where further discovery of a lower elevation is *only* available via *diagonal* correction, leading to the already-mentioned concept of increasing the *n*-coefficients-*at-once* combinatorial complexity of VCGW to cater for the greater *multidimensional acute zig-zagging/diagonality* in the model's terrain.

With respect to the terrain's shape – presume elevation at following step numbers to be in descending order: 0, 2, 5

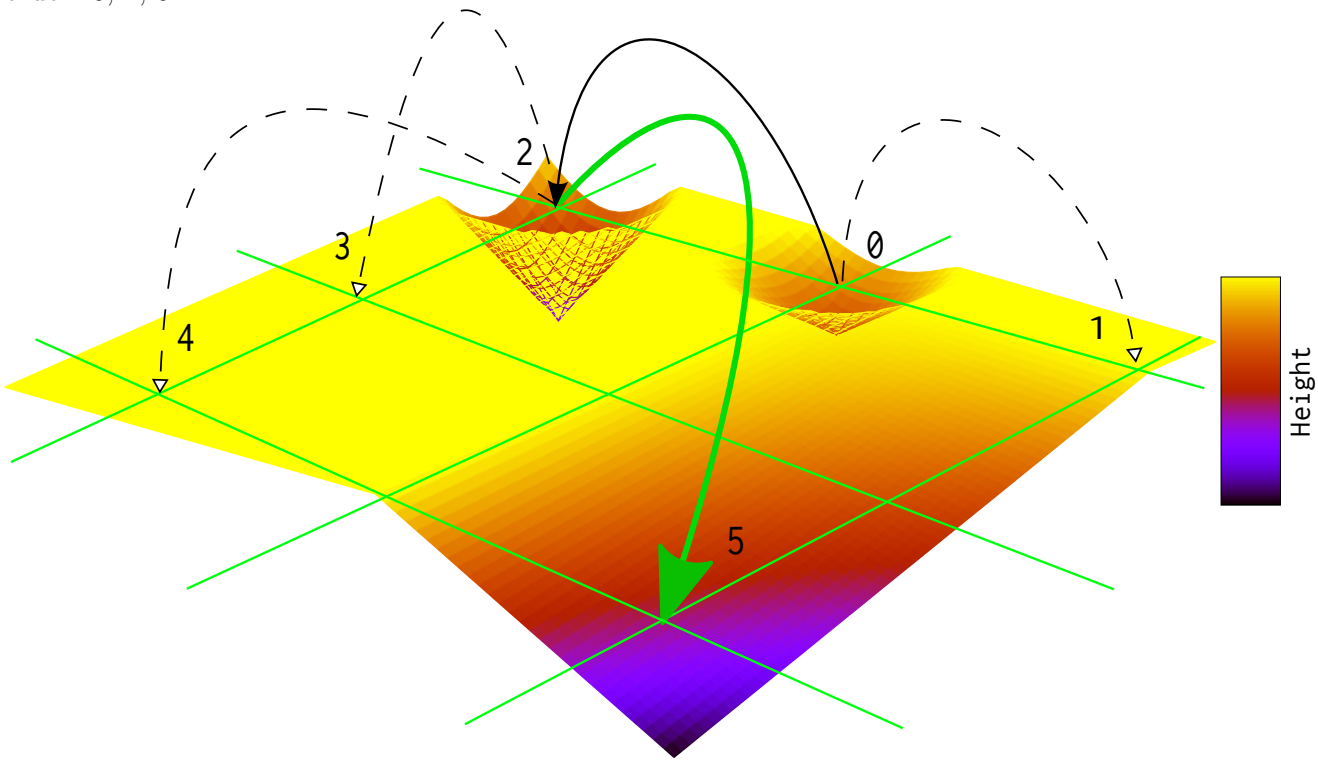


Figure 7: Diagonality traits in a given terrain implying the need to escalate from one-at-a-time complexity to *2*-coefficients-at-once modulations

2.5.3 Grid-based spacing of the sampling points is preferred over a random-based spreading of sampling locations

As has been mentioned, the terrain's exploratory area is divided into an *evenly-spaced grid* of sampling points – as opposed to picking candidate sampling points at *random* from the boundary-constrained ranges.

When compared to random-based mechanisms for picking sampling locations, the grid-like mesh exhibits more efficiency in terms of covering a given range/area as *evenly as possible* and with *as few of the grid-points as possible* thereby leading to a greater computational efficiency by evaluating lesser number of sampling locations.

For instance, if one was to assume the following

- a given model has only one dimension;
- the smallest width fluctuation in the landscape (i.e. the narrowest of troughs) which needs to be detected allows for 1 unit of measure coarseness/granularity along the dimension (e.g. similar to the aforementioned concept of sample-rate requirements from the Nyquist theory in the DSP domain);
- boundary-constrained range of exploration is from 1 to 10;

then one would have to sample the landscape with the following coordinate values:
{1, 2, 3, 4, 5, 6, 7, 8, 9, 10}

When picking a number from the aforementioned set *at random* one can easily appreciate that it may take *more* than 10 draws to obtain every member from the set – in a sequence of randomly-chosen numbers some values may occur more than once whilst others not at all:
{1, 3, 2, 3, 5, 6, 7, 5, 8, 9}

Moreover, if not quantizing the random-based sampling to integers – values drawn at random may be real numbers such as 3.2503, 1.5, 5, 7.898 etc. – then 10 random draws may yield a finer-resolution at some ranges, at the expense of coarser precision in other areas thereby potentially "missing" the narrowest of troughs:
{1.9, 2.3, 3, 1.5, 8, 8.9, 7, 6, 5.9, 7.1}
where the distance between 5.9 and 6 is narrower than it needs to be yet the gap between 3 and 5.9 is of far grater width than 1 unit of measure.

In other words, any of the aforementioned random-based approaches essentially result in a "lack of attention", or "precision-holes", in some areas of the terrain.

Of course, one is able to eventually "cover the precision-holes" simply by increasing the number of random draws. This, however, is suboptimal with respect to the efficiency of the whole process (more sampling points being evaluated means longer computational times).

Moreover, the only arguably positive characteristic of random-based draws – i.e. the finer-resolution in some areas of the terrain – is already addressed by the *zooming-in* feature of VCGW as the finesse of grid-based sampling precision increases as a "matter of course", per normal traits of the convergence process.

On a side-note, one could mention that Halton sequences also tend to provide a more even spread-of-generated-numbers when compared to random-based alternatives. It is possible to point out, however, that Halton sequences (when compared to a grid-based positioning of sampling points) exhibit lesser optimization in this particular context because a Halton sequence is designed with additional requirements and generalizations. Namely, those are:

- Halton sequence has no awareness of the "window" length of the set of draws. In other words, Halton sequence attempts to generate evenly distributed set of numbers *without* the explicit priory knowledge of the fixed number of draws that may be used in a given computational process.
- Halton sequence not only needs to produce as even coverage of generated-numbers as possible but it also needs to generate very little correlation between any of it's segments (i.e. it needs to *combine* the traits of the even coverage *with* the traits of lower correlation of random-like sequences).

None of the aforementioned issues/burdens are applicable when evaluating combinatorial complexities in VCGW and therefore a much simpler grid-like spacing of the sampling points also proves to be the most efficient one.

2.6 Final juxtaposition and re-emphasis

Of course, given the highly improbable exhaustion of *all-at-once* complexities in most of the practical implementations of the algorithm, one could point out that an acutely-diagonal terrain in a heavily multidimensional model may not be properly traversed by VCGW – whereas BHHH et al. would *apparently* perform quite well under such conditions.

This however is not the case, for in addition to the acute diagonality (e.g. zig-zagging valley) there is also a highly probable presence of a *multimodal-modulation* in the landscape's elevation – i.e. an "up and down" wave along the trough of the zig-zagging valley. Figure 8 p.24 attemps to illustrate the aforementioned point albeit in a lower-dimensional (3d) shape due to the difficulty of visualizing shapes similar to 30-dimensional diagonals.

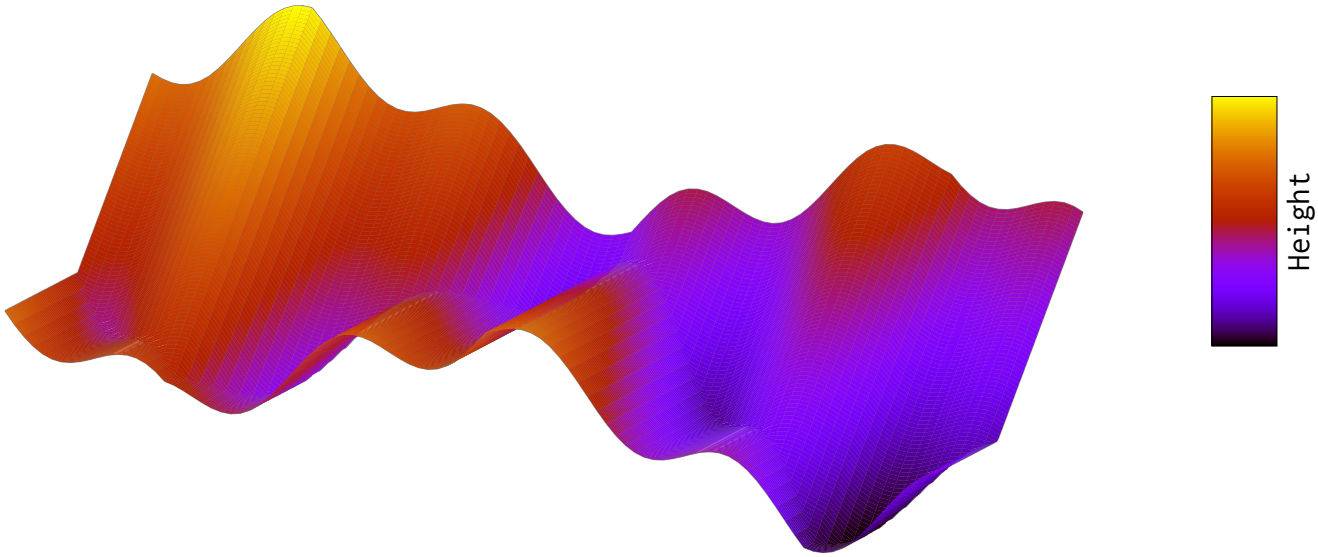


Figure 8: Zig-zagging valley in multimodal terrain where both BHHH as well as VCGW would fail (had the complexity of the shape been raised to 30 dimensions)

This brings up a previously emphasized point of not trying to expect a "silver bullet" solution from the presented algorithm as there are cases where *all*: BHHH et al. *and* VCGW would be *unable* to faithfully and robustly explore the terrain. The most extreme of such cases is, of course, a multidimensional noise/randomness as per fig 9 p.25.

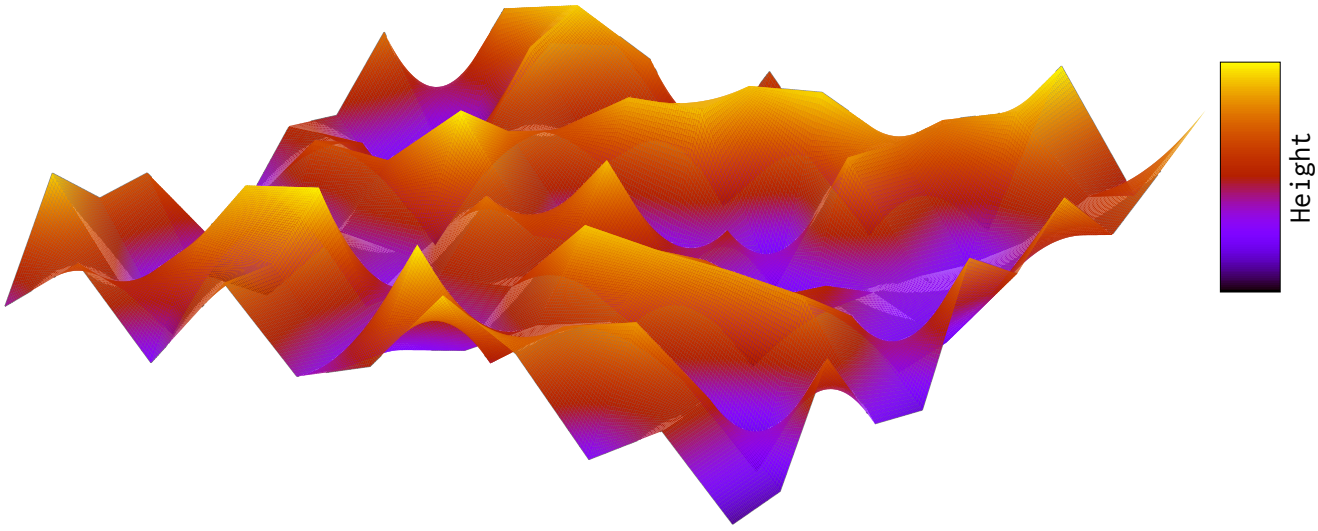


Figure 9: Random noise – an extreme example of the unsolvable terrain (once again, especially if the complexity of the shape rises to 30 dimensions)

One additional point to be made with respect to VCGW is that it's boundary-constraints may *apparently* "cull" the exploratory space quite *prematurely*.

This, perhaps, is most evident in cases of a unimodal terrain where VCGW could be prevented from "walking" down the valley (towards the lower elevation levels) simply because the algorithm's boundary-constraints may not allow it to walk any further. Comparatively, the algorithms such as BHHH et al. would have no problem in traversing such a terrain given that they do not have any explicit boundary-constraints in the first place.

This, however, is only an *apparent* problem. The solution in VCGW's case is to simply note the final, converged-on, location: if it happens to be *close-to or directly-on* the boundary then all that is needed is to augment relevant boundary constraints and redo the whole of the convergence run – keeping in mind, of course, that repeated convergence runs are already a part of normal deployment of VCGW: namely, a quality-assurance process when testing whether the converged-on location represents the dominant mode.

3 Underlying "parallel processing" infrastructure and selected notes on various implementation details

3.1 NetCPU – a Parallel computational infrastructure

As VCGW is particularly open to leveraging parallel computations – essentially, each of the to-be-evaluated sample locations can be computed independently from others – it follows that the description of the parallel computation infrastructure is but a natural step in the process of describing VCGW: not only in theoretical but also in practical, implementation-wise, terms.

3.1.1 Decoupled architecture

The aforementioned infrastructure (i.e. NetCPU) is fundamentally a decoupled architecture with one "server" and, potentially, multiple computational nodes/"workers" where:

- the server is effectively the main portion, status quo, of the algorithm as it does the coordination and maintenance of the converging process, including (but not limited to):
  - declaring the current location (point of reference) and deciding when to walk to a new location;
  - housekeeping of the remaining sampling locations (i.e. those that are still in need of being evaluated);
  - redistribution/communication of "to be evaluated" locations (specific to each particular working node);
  - controlling currently iterated-through x-at-once complexity and it's escalation, if need be, to a different x-at-once level;
  - boundary-constraints specifications, zoom-in and termination decisions;
- computational/processing nodes, on the other hand, concern themselves mainly with the task of *evaluating* various terrain locations (which are provided to each of the relevant nodes by the server); and reporting the outcomes back to the server (e.g. which sampling locations were evaluated, whether a better evaluation was found, etc.)

The decoupling capacity of the aforementioned architecture is virtually limitless:

- both: server and workers may be located on the same computer;
- server and workers may be running on different clusters in a pseudo-supercomputer environment;
- the architecture could be deployed in a LAN (small business) environment, running on day-to-day office computers;
- the whole of the Internet environment could be used where various nodes are located in geographically-disparate regions whilst participating on a semantically single task of convergence in a given exploratory terrain;
- mixture of any of the above is equally feasible.

3.1.2 CPU-bound problem domain and subsequently expanded scalability

One of the NetCPU's principal intentions is to address *cpu*-bound problems – difficulties associated with an exceptionally prolonged computations and elaborate number-crunching arithmetic – as opposed to solving *io*-bound problems where issues such as latency (delay in communication of messages between parties) and bandwidth (ability to communicate large amounts of data between devices) are of concern.

The, relative, independence from io-bound problems allows NetCPU to further expand it's scalability when considering integration of working nodes connected to *low-bandwidth or cost-sensitive* networks.

There might be a NetCPU deployment scenario where a particularly quick/instantaneous communication between infrastructure nodes (e.g. server to workers and vice versa) is simply unnecessary (even if such communication includes messages denoting a better find: a location with lower-than-current elevation). Such a scenario could easily take place when NetCPU is deployed to solve problems that take multiple hours/days to process – trying to arrange for millisecond latencies between nodes will not make much difference as the bulk of the consumed time shall reside with the act of evaluating given sampling locations, where savings on a milliseconds-scale will be dwarfed by the time taken to evaluate potentially millions of different sampling locations.

Such an observation becomes even more prominent when considering that various nodes are usually given a whole set/range of points-to-be-evaluated in a single data-compressed message thereby yielding, statistically, the vast majority of time being spent on *computing-without-communication*.

The severity of, somewhat minimal, delay in communication of a better-find (and subsequent walking to a better location) is also diminished by realization that if better-finds were so very frequent then the convergence would be completed in no time. Instead, one feels compelled to point out that a far more likely scenario (when *observed holistically across the entire lifespan of a convergence run*) would be for the better-finds to occur with a far greater rarity (especially as the algorithm progresses towards it's final termination criteria or to a greater x-at-once combinatorial complexity).

Conversely, as far as the *beneficial* outcomes of somewhat-postponed and aggregated messaging are concerned, being able to deploy additional computational nodes on slow or cost-sensitive networks – the nodes which would otherwise be unavailable as frequent message-passing et al. could flood/deteriorate the underlying network conditions – allows one to leverage even more computational resources, and precisely in the area of concern: the *cpu*-bound number-crunching.

Consequently, to reflect the aforementioned considerations, the communication procedure between the workers and the server exhibits a data *aggregation-before-reporting* behavior:

- *Working nodes deploy a periodic "reporting pulse" mechanism.*

  Processing worker keeps crunching the numbers (evaluating sampling locations) and accumulating relevant statistics: which sampling locations have been evaluated; whether a better find was obtained and if so what was it's elevation value; and so on. When a reporting pulse takes place, all of the accumulated statistics are cleared but not before being packaged in a "report" message and sent to the server.

- *The server deploys a periodic "sync/update pulse" mechanism.*

  As each of the working nodes reports it's statistics, the server accumulates arriving data in a "todo" pile. When an update pulse takes place, the server processes all of the accumulated "todo" data (such as deducting all of the reported sampled locations from the set of remaining locations in need of exploration; deciding on whether to move to a better-found location if such is present in the aggregated reported data; should a zoom-in take place; etc. – essentially updating the "state" of the convergence model). This addresses the aforementioned architecture traits:

  - *Reduction in bandwidth.*

    Each of the workers' reports, received by the server, has the capacity to trigger a communication back to the various nodes (e.g. in a form of redistributed/updated sets of to-be-evaluated-locations; updated point of reference; change in combinatorial complexity; change in a zoom-level; etc.). Responding directly to every incoming report (fired "at will" by a multitude of simultaneously-connected/active workers) without aggregation is

therefore suboptimal as it could easily produce an "unfriendly" amount of network-traffic throughout the underlying infrastructure and all the way to the network card of each of the working nodes.

As it shall be elaborated upon later, the working nodes are envisaged to be working in a "stealth" environment, both: in terms of the cpu-load as well as the io-load with respect to other activities potentially conducted on the working nodes (e.g. office-workers downloading various documents, printing reports to network printers, and so on). Consequently, any of the "unwanted" network-traffic ought to be avoided or reduced.

Moreover, quite possibly, the traffic may also become frequently redundant. A message, denoting a newly-updated state of the convergence run, may be received by the given worker only to be overridden by a new one milliseconds later – much earlier than a computational evaluation of a given sampling point may take (i.e. the temporal effect of a given message may potentially be reduced to being practically infeasible).

This further reinforces the thesis of aggregated, pulse-based processing of the workers' reports.

- *Improved scalability – allowing the server to handle a much greater number of simultaneously connected nodes.*

  There are certain CPU-bound tasks performed by the server – culling of the remaining sampling-locations in absolute (large-number software-calculated) coordinates (as discussed in further detail elsewhere in this text); decisions to reset various complexity levels; walking to new points of references; redistributing various sets of sampled-locations to multiple nodes, etc. – which would invariably become a "CPU-resources bottleneck" should the server have to compute such tasks in response to every single report arriving from every single worker. A flood of reports becomes quite feasible when considering that there may be thousands of working nodes connected to the server in a given infrastructure deployment scenario.

  It naturally follows that in order to optimize server's capability to scale-up and accommodate a multitude of working nodes it would be beneficial to migrate as many report-processing efforts as possible into an aggregated, independent from any given individual working node, computational stage which is dependent only on the frequency of the update-pulse and not on the frequency of the messages being fired at the server by potentially thousands of processing nodes. In such a case, certain processing will occur only at constant time-intervals (and only if need be: i.e. if there is some data to process in the first place) – no matter how many workers there are or how frequently their reports are arriving.

Of course, the aforementioned reporting/updating pulses could easily be made adjustable in terms of their periodicity (i.e. how frequently they take place) thereby allowing one to *tune* NetCPU's deployment over the networks which *do* tolerate higher io-load and therefore giving one a chance to easily optimize the performance improvements in messaging should the network tolerate greater traffic-flooding. Moreover, such an optimization comes virtually for free where simply adjusting the frequency of the pulse results in the ability to more comprehensively utilize the network-capacity of a given deployment scenario.

3.1.3 Stealth mode – prioritized scheduling and harnessing small chunks of available CPU power across many nodes

NetCPU is designed to be non-intrusive with respect to both: CPU-bound as well as io-bound activities. The term *non-intrusive* is used here with respect to other, irrelevant to NetCPU, activities which could potentially take place on the working nodes. A given working node, for example, could represent an office computer used by company's personnel for activities such as Internet-browsing, email, printing, report-writing and so on. In fact, given the size of a generic organization one can easily appreciate the multitude of such computers being present in a given office environment.

One of the natural progressions from the aforementioned example is to realize that a given computer's CPU may not be used 100% of the time to it's full capacity when looked-at from a deep-enough level of technical analysis (as illustrated in figure 10, p.29).

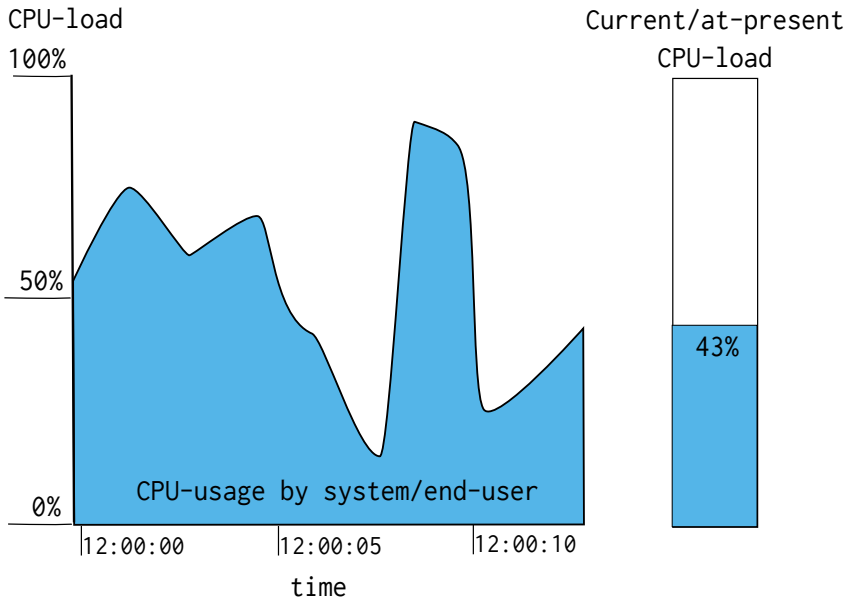


Figure 10: Generic CPU-activity graph

What may seem as a continuous activity of word-processing or email-communication is actually a process percolated with multi-millisecond -second -minute chunks of inactivity when a computer is underutilized with respect to it's full number-crunching capacity. Such chunks-of-inactivity could be caused by user temporarily pausing page-scrolling in an Internet browser, millisecond pauses between keystrokes on the keyboard, taking a few seconds to compose one's thoughts when writing a paragraph of text or, indeed, simply having a cup of tea. One is able to make a further observation stating that many of modern computers have a multi-core CPU architecture (where there are *multiple* simultaneously-running number-crunching modules) – thereby yielding an even higher number of occurrences when CPU is either at rest or is being heavily underutilized.

The principal thesis of this idea is to harness as much of the aforementioned available, and otherwise unused, computational power as possible. Graphically, and from a somewhat simplified/idealistic standpoint (to facilitate the brevity of this argument), this would imply filling-in the unused area (above the previously-illustrated CPU-load curves) all the way up to the 100% mark so that the resulting CPU-load utilization shows full-throttle usage.

In figure 11, p.30 the resulting CPU-usage yields a full 100% where the "full-throttle" is comprised from whatever portion of the CPU-power the user-activity needs (e.g. scrolling a webpage in an Internet browser); and the remainder being scavanged by the NetCPU computational infrastructure.

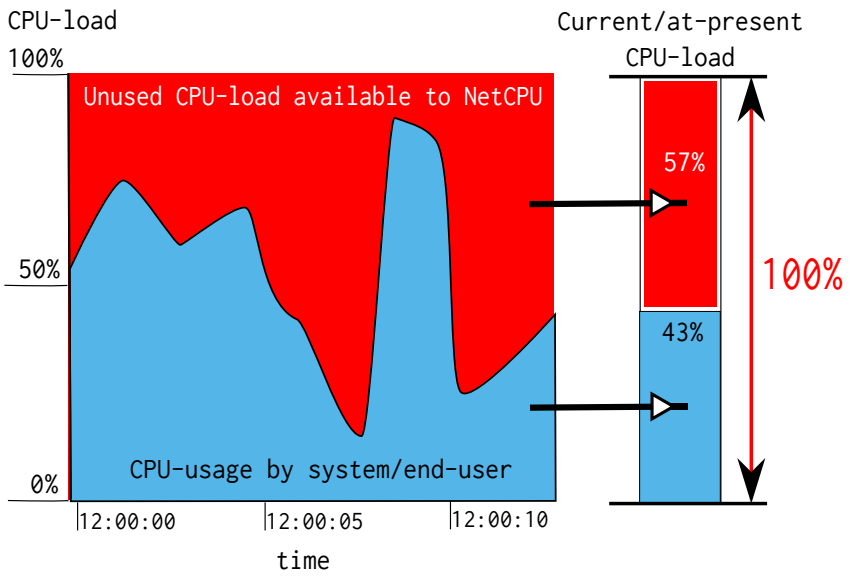


Figure 11: Fully optimized CPU-activity graph

In practice, a process of harnessing absolutely all of the unused CPU-power at any given moment is somewhat approximated due to varying degrees of sophistication in the scheduling and multi-tasking of programs exhibited by different platforms (e.g. MS Windows, Linux, Mac OSX, FreeBSD and so on). Essentially, however, such a process is achieved via a *priority*-based scheduling of computational threads.

In layman's terms, NetCPU infrastructure is treated as a "second-class" citizen – should there be any other program requiring computational resources (e.g. email, word-processor, etc.) the underlying system would suspend any of the NetCPU's activities and run the other program(s) instead.

An even simpler example is that of a priority-based seating on a bus. If there is an empty seat then any generic passenger can occupy it. If, however, an elderly person comes in, then the already-seating, unprivileged, occupant would need to vacate the seat for the elderly passenger. In such terms, NetCPU is an unprivileged occupant and any other program/activity running on a given computer is a prioritized, elderly, passenger.

Returning, briefly, to the point of different systems being unequal in their sophistication when it comes to priority-based scheduling of different tasks, one ought to mention that there are indeed cases where the underlying system may not properly prioritize NetCPU computations as being low-enough so as not to interfere with other, higher-priority tasks. For instance, there have been empirically established cases where a highest-priority end-user task was slowed down by as much as 30% when running together with a lowest-possible priority CPU-crunching thread. The reasons for such a situation are rather technical and vary from system to system (e.g. the higher-priority task performs a lot of memory allocation/de-allocation resulting in "soft" page-faults which affects how the system's scheduler runs other "ready-to-run" albeit already lowest-priority processes and thereby affecting when the higher-priority task is given a chance to run again on the next scheduling/interrupt cycle;

and so on, and so on indeed).

To this extent, on relevant platforms/systems, NetCPU takes explicit and proactive steps to further *disadvantage itself*. Once again, the technical details are rather tedious and somewhat outside the scope of this document – e.g. running on a fixed CPU-affinity; utilizing periodic short sleeps/cat-naps to detect and measure if there are any other tasks which had utilized CPU above a certain threshold and, if so, sleeping/backing-off for an even longer period of time; etc. etc. etc. – it shall suffice, however, to state in laymen terms that the lack of sophistication in the default system's scheduling mechanisms is not visited in a form of disposition onto the end-user tasks but, rather, it results in further *detriment to the NetCPU's activities* whose computations are being postponed/disadvantaged to a greater degree (i.e. having an even lesser chance to perform calculations).

Naturally, the overall stealth approach (be it due to a default, sophisticated-enough, lowest-priority scheduling or explicit steps towards further computational detriment) leaves NetCPU with a plethora of computational pauses and interruptions. It must be mentioned, however, that such a scenario is intentional and represents "something is better than nothing" concept.

Given a traditional office environment, it would be practically infeasible to overload computers' CPUs to such an extent that they become non-responsive to the end-users. Instead, computers alternate in wearing multiple "hats" – normally they wear "end-user tasks hat" and whenever there is an opportunity to switch, albeit momentarily, computers put on a "NetCPU hat" and do their little bit of computational participation. In other words, running in stealth mode and taking advantage of small chunks of the unused CPU power is far better than running in a full-foreground, "take-over-the-computer", manner only to witness the number of available computers reduced dramatically as the majority of end-users would simply refuse to allow NetCPU to be deployed on their machines – on the grounds of their computers becoming unworkable.

Moreover, looking at the total of the underutilized computational power being present at any given time in a generic office environment (i.e. calculated cumulatively across all of the office computers et al.) one begins to appreciate the formidable capability of such an arrangement. In other words – something is better than nothing and especially so when "something" is quite a lot to begin with.

Additionally, the aforementioned approach becomes even more powerful during the end-of-business hours when office workers (i.e. end-users) go home thereby allowing NetCPU to run uninterrupted for much longer periods.

The Internet-scalable nature of NetCPU also allows one to leverage different timezones of the participating institutions where end-of-business hours do not take place simultaneously for organizations located in disparate geographical regions. This allows for a more continuous/smooth functionality of the overall NetCPU deployment where "during the business-hours" intermittent computational pauses from one organization are compensated with more continuous "afther the business hours" computations from another organization.

Naturally, priority-based scheduling can be fully parametric/adjustable – should one envisage a deployment scenario where some of the working nodes are available for NetCPU's *exclusive* use (e.g. old/re-intregrated/recycled computers) then it should be possible to escalate the scheduling-priority of NetCPU's computations thereby taking, unconditionally, all of computers' resources (e.g. close to real-time scheduling priority). In other words, ability to control NetCPU's scheduling priority allows for a greater efficiency in the deployment of the available computational resources when preparing a collection of working nodes for participation in the NetCPU's infrastructure.

As it was most likely mentioned already – the stealth mode is characterized not only with respect to

harnessing available/unused CPU power, but also in relation to being non-intrusive in other aspects of computer activity such as network traffic, hard-disk activity and so on. For example, office personnel may be trying to communicate valuable information across devices in a timely manner and so the bandwidth capacity of the underlying network becomes of paramount importance. To this extent the aforementioned bandwidth-economy -sensitivity of the NetCPU addresses this issue via compressed data messages, utilization of smaller-data representation of absolute coordinates when being communicated between computational nodes and so on.

3.1.4 Fault-tolerance and runtime robustness of the holistic computational process

Given a somewhat *ad hoc* nature of working-nodes' participation in the overall NetCPU-convergence – e.g. some computers may be turned off by their users; others may go on-line and become available for NetCPU participation half-way through a given convergence run; moreover, certain computational nodes may drastically slow-down their computational progress due to the external-to-NetCPU activities conducted by the end-users, etc. – it is of importance to note NetCPU's fault-tolerance capabilities.

In other words, despite whatever may or may not happen on any given working node – the overall convergence run in a given NetCPU infrastructure shall not be terminated. It may be slowed-down (due to a lesser number of active/working participants, etc.) but the process as a whole shall not be put in jeopardy of being restarted from scratch.

Partly this is achieved by the way of *job re-servicing* – if a given working node is not evaluating it's share of sampling locations (or is doing so very slowly) whilst other nodes have done evaluating their own portions of calculations, then the server is able to use such nodes-without-any-pending-evaluations to evaluate any of the remaining sampling locations on behalf of the missing/slow/"lazy" node. This acts as an implicit load-balancing mechanism where faster/more-powerful nodes will perform more computations as compared to their slower counterparts.

Given the aforementioned re-servicing of jobs to various nodes, there is a distinct possibility that at some point in time different nodes may have identical sampling locations in their "todo pile". To this extent, when a given node reports a set of evaluated locations to the server, the server determines if there are any other nodes which are still tasked with performing the same evaluations and, if so, re-distributes the updated set of sampling locations (i.e. without the already-done locations) to all of the relevant workers. This allows working nodes, upon the receipt of the updated set of sampling locations, to cull any of the "no-longer-necessary" sampling locations from their "todo" piles.

Moreover, to address the possibility of downtime on the server-side (e.g. power-outage, etc.), the server saves it's state-of-convergence so that it would able to re-start from the last saved "sync point" should the power outage (or similar issue) take place thereby avoiding previously mentioned restart of the whole of the convergence run from scratch.

3.1.5 Overall traits of the NetCPU infrastructure

At this stage, it might be of use to list some of the general NetCPU characteristics thereby helping one to understand as to why it was created in the first place – as opposed to deploying some of the already-existing, 3rd-party parallel-processing frameworks:

- SSL (peer-to-peer encrypted) communication utilizing both: server-to-client as well as client(s)-

to-server, certificate-based authentication – allowing for the processing of private, sensitive data in an Internet-wide computational environment.

- Certificate-based site/group-of-computers authorization/rejection – catering for administrative issues associated with formation of partnerships between various departments and organizations for the purpose of calculating models of common interest.
- Low-bandwidth utilization (option to prioritize smaller data over lower latency).
- Ad-hoc discovery, connectivity and the overall dynamic participation-topology.
- Implicit fault-tolerance (processing nodes going on- off-line, stalling, etc.) with respect to the overall convergence process.
- Low CPU and memory priority-scheduling with advanced tuning (as default, OS-provided API for such purposes has proven to be insufficient on some systems).
- End-user specialization overrides (e.g. end-user being able to decide which cores/CPU units are not to be used by the infrastructure, etc.)
- Automatic update and redistribution of relevant binaries and applications. This applies not only to the updated, numeric-processing executables but also to the plethora of helper utilities such as hybernation-controllers based on CPU and IO activities for the host computers – as was shown to be of need for some of the systems.
- Processing-worker's (i.e. the number-crunching application) ability to automatically opt-out of a given processing task due to host's resources capacity, yet remaining on "stand-by" for the participation in the future, potentially applicable, tasks.
- Support for simultaneous mix of multi-platform (OS-wise) and multi-architecture (e.g. 32/64 bit, etc.) environments.
- Messaging-protocol decoupling and extensibility where different computational participants may be implemented via various programming languages and deploying different technologies (e.g. OpenCL, multi- or single-threaded IPC design, etc. etc.)
- Tighter integration with the deployed convergence algorithms and models.

3.2 On the implementation of the "previosly-sampled locations" memory

As mentioned earlier, all of the previously-sampled locations are remembered by VCGW in a semantically *absolute* – independent from any given point-of-reference on the walking path – coordinates.

To elaborate on this point a little further consider the following, albeit rather exaggerated in some aspects and simplified in others, example:

- the exploratory terrain is divided into a grid of 2 sampling points per every dimension;
- total number of dimensions/coefficients in a given terrain is 77;
- the highest to-be-explored combinatorial complexity is *one-at-a-time* (i.e. $n = 1$ in n-coefficients-*at-once*);

the algorithm then has a total of 77 different sample-points to evaluate (with respect to current location-of-reference). Such a number could easily be represented by an integer native to most of the existing computational hardware (e.g. an 8-bit variable i where $i < 256$).

However, such considerations only hold true with respect to *current location of reference*. When the algorithm walks to a *different* location, there would be a need to represent *previously* sampled places (from the old locations-of-reference) and correlate them with the *about-to-be-sampled* places in relation to the new location-of-reference – so that the algorithm may see if some of the future sampling locations have already been evaluated. If previously sampled places were only represented with respect to their own, old locations of reference, then the aforementioned correlation would not be possible.

In other words, an *absolute* coordinate system (i.e. independent from any given point of reference) needs to be deployed.

From a practical standpoint, when applied to example at hand (i.e. a total of 77 dimensions in the model) this yields a presence of a very large range – up to 2^{77} different sampling locations. Naturally, this is unlikely to be representable by native, hardware-based, arithmetic and consequently software-centric mathematical libraries need to be used (e.g. GNU's MP BIGNUM et al.).

From the standpoint of computational-efficiency one feels behooved to point out, however, that it is *only* the mapping, and subsequent culling of previously sampled locations, that are done in large-number, slow mathematical computations. The *frequently-deployed* sample-point coordinate utilization – when iterating through x-at-once combinatorial complexities and evaluating various sampling points – are all calculated with respect to the given location-of-reference (i.e. hardware-native numeric formats).

So, in other words, there are *two* contexts for keeping the coordinates of sampled locations:

- hardwarde-native computations for the total *explorable* complexity (e.g. covering the range of up to and including n-at-once combinations in a N-dimensional model where $n <= N$) – used in deriving coefficients' values when composing points' coordinates and subsequently evaluating their elevation values; and
- software-based calculations for astronomically large numbers (e.g. in terms of absolute N-at-once, all-of-coefficients coordinates) – used only for comparative mapping and culling of the previously sampled locations.

As to the question of why not implement previously-sampled-locations memory as an explicit array of coefficient values (as opposed to the use of something like GNU's MP BIGNUM et al.) the following rationale could be mentioned:

- Each coefficient would then be stored as a floating-point variable. Whilst the aforementioned method of accounting for absolute coordinates is able to represent any value as a single 77-bit integer, the 77 floating-point numbers – each comprising of 4 or 8 bytes – would yield at least 32 times greater memory consumption.

  This is clearly suboptimal considering that even a modest parameterization of the algorithm such as 5 grid-points, 4-at-once complexity in a 28-dimensional terrain results in a very large number of all possible combinations of up to any four coefficients from a total pool of 28 coefficients.

Instead, an index-based accounting of absolute coordinates (using large integers via software-implemented math such as BIGNUM) is used where a given location's index is modulated via predetermined transformations to/from the explicit, floating-point, coefficient values. In other words, every possible permutation of every grid-point in *all*-coefficients (y-dimensional model) context is effectively given a unique index value which is then used to unambiguously translate such an index to an explicit set of dimensional coordinates. Moreover, such indexed locations are coalesced into continuous ranges where possible – e.g. a sequence of *indexed* sampling locations such as {
10000000000000000000000000000000000000000000000000000000000000005,
10000000000000000000000000000000000000000000000000000000000000006,
10000000000000000000000000000000000000000000000000000000000000007,
10000000000000000000000000000000000000000000000000000000000000008,
10000000000000000000000000000000000000000000000000000000000000009,
10000000000000000000000000000000000000000000000000000000000000010
} would be replaced by a pair of numbers {
10000000000000000000000000000000000000000000000000000000000000005,
6
} denoting the starting location and the size of the region.

- Floating-point calculations and comparisons done on different hardware architectures – a distinct possibility in a parallel computational environment where each of the computing nodes may posses different floating-point registers architecture – may not be exactly the same when computed and communicated on different machines (e.g. 10/3 on one box is not the same as 10/3 on another – when being compared for explicit equality).

  Comparing for near-enough (rounded) equalities would require one to take additional care regarding the establishment of what exactly is near-enough and what is not based on differing hardware precisions/rounding-modes/etc. (potentially leading to some locations being evaluated unnecessarily more often by different nodes); and how it correlates to a current zoom-level and fineness of the boundary-constraints of the algorithm (e.g. model-specifications yielding the need to explore the terrain on microscopic levels of scale).

  Communicating indexed integral values instead allows for a much greater consistency with respect to determining which locations have been already done and which ought to remain in the "todo" pile, irrespective of the underlying computational-node's architecture.

Naturally, one is not saying that it would be impossible to implement explicit, floating-point, coefficient-wise storage for the previously-sampled locations – only that it may be suboptimal when compared to a large-integer approach of communication, comparison and re-distribution of coordinates across different nodes. This stands to reason with respect to both: the memory consumption and the additional floating-point arithmetic when comparing, coalescing and culling the ranges of various locations.

4 VCGW-related model-specific optimization example

What occurs quite frequently in most of the VCGW's deployment scenarios is that *fewer* than all-at-once coefficients are modified between *successive* invocations of the model's evaluation routine (i.e. the routine responsible for yielding terrain's elevation value for a given location/set-of-coefficients-values). This stands true for any x-at-once combinatorial complexity in a N-dimensional model, where $x < N$.

For example, when a mixed-logit model is being evaluated by VCGW, any two or more consecutive evaluations of the model's likelihood function will probably see *some coefficients remain unchanged*. Such an observation allows one to leverage a rather specific optimization, with respect to computational performance, by way of "caching/saving" certain equation-portions from previous invocations of the evaluation routine – so that such portions may be re-used later-on during subsequent likelihood evaluations (without having to re-compute the same/unchanging equation-portions over and over again).

To illustrate this point a little further, consider a mathematical expression for the likelihood of a generic mixed logit model:

$$P_j = \frac{e^{(\boldsymbol{\beta}+\boldsymbol{\eta})\boldsymbol{x}_j}}{\sum_{a=1}^{A} e^{(\boldsymbol{\beta}+\boldsymbol{\eta})\boldsymbol{x}_a}} \tag{1}$$

where

- A is the total number of alternatives
- $\boldsymbol{x}_a$ is the observation vector denoting a choice of attributes for alternative a, such that $1 <= a <= A$
- P_j is the probability of alternative j being chosen where, once again, $1 <= j <= A$
- $\boldsymbol{\beta}$ is the vector of attribute-weights
- $\boldsymbol{\eta}$ is the vector of attribute-related variances

From equation 1, the $e^{(\boldsymbol{\beta}+\boldsymbol{\eta})\boldsymbol{x}_j}$ portion could be extrapolated to include attribute-specific decompositions of $\boldsymbol{\beta} = (\beta_1, \ldots, \beta_n)$ and $\boldsymbol{\eta} = (\eta_1, \ldots, \eta_n)$ yielding a somewhat expanded form:

$$e^{(\beta_1+\eta_1,\ldots,\beta_n+\eta_n)\boldsymbol{x}_j} \tag{2}$$

Thereby expanding the overall mixed logit listed in equation 1 into:

$$P_j = \frac{e^{(\beta_1+\eta_1,\ldots,\beta_n+\eta_n)\boldsymbol{x}_j}}{\sum_{a=1}^{A} e^{(\beta_1+\eta_1,\ldots,\beta_n+\eta_n)\boldsymbol{x}_a}} \tag{3}$$

From the equation 3 it could be seen that if, at a given moment, only the coefficient no. 1 is being modulated – meaning that one-at-a-time combinatorial complexity is being deployed by VCGW and, currently, the coefficient no. 1 is being modulated along it's grid-points – then other coefficients are *left unchanged between successive likelihood-function evaluations*.

This implies that out of $(\beta_1 + \eta_1, \ldots, \beta_n + \eta_n)$ additions only the first one $(\beta_1 + \eta_1)$ needs to be re-calculated whilst others $(\beta_2 + \eta_2, \ldots, \beta_n + \eta_n)$ can be re-used/recycled. In other words, the likelihood

formula is broken up into sections which depend only on specific coefficients and, more importantly, completely *independent* from others.

The savings from the aforementioned recycled additions/sums may not appear to be overly dramatic until one realizes that in many implementations of mixed logit, the likelihood-evaluation involves a randomly-distributed series of "draws" for the attribute variances (i.e. η). Thusly, in a hypothetical case of 500 draws, the equation 3 shall be evaluated 500 times more thereby yielding the augmentation of the saved additions by a factor of 500 (as per table 3, p.37).

draw no.	1_{st}-coefficient math	2_{nd}-coefficient math	...	n_{th}-coefficient math
1	$\beta_1 + \eta_1^1$	$\beta_2 + \eta_2^1$	...	$\beta_n + \eta_n^1$
2	$\beta_1 + \eta_1^2$	$\beta_2 + \eta_2^2$	...	$\beta_n + \eta_n^2$
3	$\beta_1 + \eta_1^3$	$\beta_2 + \eta_2^3$	...	$\beta_n + \eta_n^3$
...	$\beta_1 + \eta_1^{(...)}$	$\beta_2 + \eta_2^{(...)}$	...	$\beta_n + \eta_n^{(...)}$
499	$\beta_1 + \eta_1^{499}$	$\beta_2 + \eta_2^{499}$	...	$\beta_n + \eta_n^{499}$
500	$\beta_1 + \eta_1^{500}$	$\beta_2 + \eta_2^{500}$	...	$\beta_n + \eta_n^{500}$

Table 3: Cached/recycled calculations (highlighted in green) in a 500-draws scenario where only the 1^{st} coefficient is being modified.
Note with respect to coefficient-wise notation:

- superscripts denote the draws
- subscripts indicate coefficients

Naturally, the attribute-based decomposition, exemplified in equation 2, could be expanded even further with respect to multiplication by the observation vector x_j, yielding:

$$e^{(\beta_1+\eta_1,...,\beta_n+\eta_n)\boldsymbol{x}_j} = e^{[(\beta_1+\eta_1)x_{j1}+,...,+(\beta_n+\eta_n)x_{jn}]} \tag{4}$$

However, when considering that in most datasets/surveys evaluated via mixed logit models there are numerous:

- alternatives per choice;
- choice-observations per respondent;
- respondents;

the questions regarding RAM availability/capacity become of priority as each of the variations in the equation 4 with respect to different x_j values et. al. would need to be cached/remembered (thereby occupying more memory for their storage).

The RAM utilization becomes even more important when realizing that:

- NetCPU infrastructure is running in stealth-mode and it would therefore be infeasible to overtax the overall system memory of various computational nodes.

- The decoupled nature of NetCPU allows it to run on a plethora of different hardware architectures with varying RAM capacities and so a given deployment of a given model must take this into consideration.

Naturally, one is able to generate different, more specialized, implementations of the same (e.g. mixed logit) model where each is deployed on a more-appropriate architecture (e.g. some with greater RAM capacity/utilization, some with more economic view of the memory's availability). The aforementioned examples are therefore just that: non-exhaustive examples of various possibilities which may use the nature of the *x-at-once* combinatorial complexity in VCGW algorithm in order to further optimize computational efficiency of the calculated models.

5 Document version description

Analyzing this document's version (e.g. 0.0.1.1) the digits *from-right-to-left* indicate the following:

- the right-most digit is for semantically-insignificant changes such as typos, spelling, etc.
- the next digit is for additions/subtractions related to further clarifications and explanations which do not change the nature of the algorithm/work. For example, at some point such a digit was changed from 0 to 1 because a small section was added outlining a set of overall reasons for developing NetCPU infrastructure as opposed to using/adapting various 3rd-party frameworks;
- the next digit is for changes associated with the design of the algorithm/work. Such changes however are still *backwards*-compatible (e.g. improvements, additional ideas, etc. which do not negate or change the work in relation to it's previous version);
- the last, left-most, digit is for changes that would significantly change the work to a point of negating/correcting/replacing the previous design or parts thereof.

