A Predictive Data Analytic for the Hardness of Hamiltonian Cycle Problem Instances

Gijs van Horn*, Richard Olij†, Joeri Sleegers‡ and Daan van den Berg§

* Indivirtual and University of Amsterdam, The Netherlands
† University of Amsterdam, The Netherlands
‡ Olisto and University of Amsterdam, The Netherlands
§ Docentengroep IvI, Faculty of Science, University of Amsterdam, The Netherlands

contact@gijsvanhorn.nl*, richard.olij@student.uva.nl†, joeri.sleegers@olisto.com‡, d.vandenberg@uva.nl§

Abstract*—In their landmark paper ”Where the *Really* Hard Problems Are”, Cheeseman et al. describe the relative instance hardness, measured in computation time, of three decision problems (Hamiltonian Cycle, Vertex Coloring, K-satisfiability) and one optimization problem (Traveling Salesman). For these four problems, they identify a single property, an ”order parameter” related to specific instance characteristics, for predicting computational hardness. One such characteristic is the probability of a random graph being Hamiltonian (having a Hamiltonian Cycle): it depends on its average vertex degree, which is its order parameter. This Hamiltonian probability goes through a sudden phase transition as the order parameter increases and the hardest problem instances, algorithmically speaking, are found close to this phase transition. As such, the order parameter can be seen as an analytic on instance data useful for predicting runtimes on (exponential time) algorithms. In this study, we replicate the original experiment and extend it with two more algorithms. Our countribution is as follows: first, we confirm their original results. Second, we show that an inversion of their heuristic significantly improves algorithmic performance on the same graphs, at zero extra cost. Third, we show that an advanced pruning algorithm by Vandegriend and Culberson further improves runtimes when run on the same graphs. We conclude that the order parameter based on problem instance data analytics is useful across different algorithms. Fourth, we produce high-resolution online interactive diagrams, which we make available for further research along with all the source code and input data.

***Keywords*—Hamiltonian Cycle; exact algorithm; exhaustive algorithm; heuristic; phase transition; order parameter; data analytics; instance hardness; replication.**

I. Introduction

The ”Great Divide” between P and NP has haunted computer science and related disciplines for over half a century. Problems in P are problems for which the runtime of the best known algorithm increases polynomially with the problem size, like calculating the average of an array of numbers. If the array doubles in size, so does the runtime of the best known algorithm - a polynomial increase. A problem in NP however, has no such algorithm and it is an open question whether it will ever be found. An example hereof is ”satisfiability” (sometimes abbreviated to SAT), a problem in which an algorithm assigns values ’true’ or ’false’ to variables in Boolean formulas like $(a \vee \neg b \vee d) \wedge (b \vee c \vee \neg d)$ such that the formula as a whole is satisfied (becomes ’true’), or making sure that no such assignment exists. Algorithms that do this, and are guaranteed to give a solution whenever it exists and return no otherwise, are called *complete* algorithms.

Being complete is a great virtue for an algorithm, but it comes at a hefty price. Often, these algorithms operate by *brute-force*: simply trying all combinations for all variables until a solution is found, which usually takes vast amounts of time. Smarter algorithms exist too; clever pruning can speed things up by excluding large sections of state-space, at the cost of some extra computational instructions, an investment that usually pays off. Heuristic algorithms are also fast but not necessarily complete - so it is not guaranteed a solution is found if one exists. After decades of research, runtimes of even the most efficient complete SAT-algorithm known today still increases exponentially with the number of variables – much worse than polynomial, even for low exponents. Therefore, SAT is in NP, a class of ”Notorious Problems” that rapidly become unsolvable as their size increases. In practice, this means that satisfiability problems (and other problems in NP) with only a few hundred variables are practically unsolvable, whereas industries such as chip manufacture or program verification in software engineering could typically employ millions [1] [2].

So, the problem class NP might be considered ”the class of dashed hopes and idle dreams”, but nonetheless scientists managed to pry loose a few bricks in the great wall that separates P from NP. Most notably, the seminal work ”Where the *Really* Hard Problems Are” by Cheeseman, Kanefsky and Taylor (henceforth abbreviated to ’Cetal’), showed that although runtime increases non-polynomially for NP-problems, some *instances* of these hard problems might actually be easy to solve [3]. Not every formula in SAT is hard – easily satisfiable formulas exist too, even with many variables, but the hard ones keep the problem as a whole in NP. But Cetal’s great contribution was not only to expose the huge differences in instances hardness within a single NP-problem, they also showed *where* those really hard instances are – and how to get there. Their findings were followed up numerous times and truly exposed some of the intricate inner anatomy of instance hardness, and problem class hardness as a whole.

So, where *are* these notorious hard problem instances then? According to Cetal, they are hiding in the phase transition. As their constrainedness increases, the problem instances suddenly jump from having many solutions to having no solutions. For an example in satisfiability, most randomly generated SAT-formulas of two clauses and four variables such as our formula $(a \vee \neg b \vee d) \wedge (b \vee c \vee \neg d)$ are easily satisfiable; they have many assignments that make them true. But as soon as the order parameter, the ratio of clauses versus variables α, passes 4.26, (almost) no satisfiable formulas exist [4] [5]. So, if we randomly generate a formula with 20 or more clauses on these same four variables, it is almost certainly unsatisfiable and those rare formulas that *are* satisfiable beyond the phase transition have very few solutions – which counterintuitively enough makes them easy again. So, for most complete algorithms, both extremes are quickly decided: for the highly satisfiable formulas in $\alpha << 4.26$, a solution is quickly found, and unsatisfiable formulas in $\alpha >> 4.26$ are quickly proven as such. But in between, just around $\alpha = 4.26$, where the transition from satisfiable to unsatisfiable takes place, are formulas that take the longest to decide upon. This is where the really hard problem instances are: hiding in the phase transition. But Cetal identify this order parameter not only for SAT; the Hamiltonian cycle problem (explained in detial in Section II) has one too, and so does Vertex Coloring. Again, the phase transition is where the really hard problem instances are and although their rather coarse seminal results on these problems have been followed up in more detail, they are solid [5]–[8]. Or to put it in a later quote by Ian Gent and Toby Walsh: "[Indeed, we have yet to find an NP-complete problem that *lacks* a phase transition]" [9].

In hindsight, but only in hindsight, the ubiquity of phase transitions throughout the class is not a complete surprise. Satisfiability, Vertex Coloring and the Hamiltonian cycle problem are NP-complete problems; a subset of problems in NP that with more or less effort can be transformed into each other [10]. This means a lot. This means that if someone finds a polynomial complete algorithm for just one of these problems, all of them become easy and the whole hardness class will simply evaporate. That person would also be an instant millionaire thanks to the Clay Mathematics Institute that listed the P$\overset{?}{=}$NP-question as one of their Millenium Problems [11]. But the intricate relations inside NP-completeness might also stretch into the properties of phase transitions and instance hardness. Or, to pour it into another fluid expression by Ian Gent and Toby Walsh "[Although any NP-complete problem can be transformed into any other NP-complete problem, this mapping does *not* map the problem space uniformly]" [9]. So, a phase transition in say, satisfiability, does *not* guarantee the existence of a phase transition in Hamiltonian Cycle or in Vertex Coloring. The fact is though, that Cetal do find them for all three.

In the next section, we will look at the Hamiltonian cycle problem, how it depends on the average vertex degree of a graph, and give an overview of the available algorithms for the problem so far. In Section III, we will explain the algorithm from Cetal's original experiment, and the two algorithms we've added as an extension. In Section IV, the experimental details and results are laid out in detail. In Section V, we conclude that the Cetal's findings are replicable, but also that the order parameter serves as a predictive data analytic on other algorithms. We also discuss the implications. Section VI contains acknowledgements, and a small tribute to Cetal's original work.

II. The Hamiltonian Phase Transition

The Hamiltonian cycle problem comes in many different varieties, but in its most elementary form it involves finding a path (a sequence of distinct edges) in an undirected and unweighted graph that visits every vertex exactly once, and forms a closed loop. The probability of a random graph being Hamiltonian (i.e., having a Hamiltonian Cycle), has been thoroughly studied [12]–[14]. In the limit, it is a smooth function of vertex degree and therefore the probability for a random graph of V Vertices and E edges being Hamiltonian can be calculated analytically:

$$P_{Hamiltonian}(V,E) = e^{-e^{-2c}} \quad (1)$$

in which

$$E = \frac{1}{2}\ Vln(V) + \frac{1}{2}\ Vln(ln(V)) + cV \quad (2)$$

Like the phase transition around α in SAT, the Hamiltonian phase transition is also sigmoidally shaped across a threshold point, the average degree of $ln(V) + ln(ln(V))$ for a graph of V vertices (also see Figure 1). The phase transition gets ever steeper for larger graphs, until it becomes instantaneous at the threshold point as V goes to infinity. For this (theoretical) reason, the probability of being Hamiltonian at the threshold point is somewhat below 0.5 at $e^{-1} \approx 0.368$,

The probability of being Hamiltonian is one thing, deciding whether a given graph actually *has* a Hamiltonian cycle is quite another. A great number of complete algorithms have been developed through the years, the earliest being exhaustive methods that could run in $O(n!)$ time [15]. A dynamic programming approach, quite advanced for the time, running in $O(n^2 2^n)$ was built by by Michael Held & Richard Karp, and by Richard Bellman independently [16] [17]. Some early pruning efforts can be found in the work of Silvano Martello and Frank Rubin whose algorithms could still run in $O(n!)$ but are in practice probably much faster [18] [19]. Many of their techniques eventually ended up in the algorithm by Vandegriend & Culberson (henceforth 'Vacul'), which we rebuilt as part of this replication, and can be found in Section III [20]. Algorithms by Bollobás and Björklund run faster than Bellman–Held–Karp, but are technically speaking not complete for finite graphs [21] [22]. The 2007 algorithm by Iwama & Nakashima [23] runs in $O(2^{1.251n})$ time on cubic graphs, thereby improving Eppstein's 2003 algorithm that runs

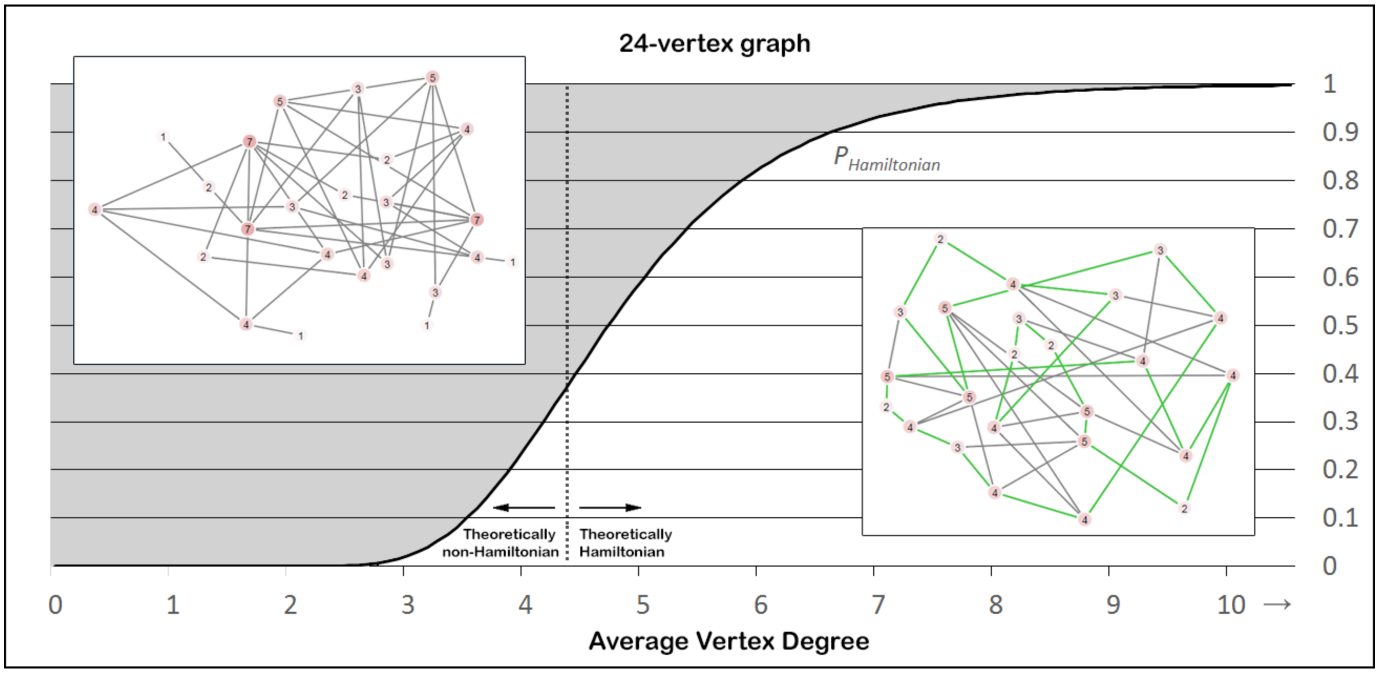

Fig. 1. The probability of a randomly generated graph being Hamiltonian depends on the average vertex degree, and is sigmoidally shaped around the threshold point of $ln(V) + ln(ln(V))$. Top-left inset is a non-Hamiltonian random graph, bottom-right inset is a Hamiltonian graph with the Hamiltonian cycle itself being highlighted.

in $O(2^{1.260n})$. While these kind of marginal improvements on specialized instances are typical for the progress in the field, these two actually deserve some extra attention.

The cubic graph, in which every vertex has a maximum degree of three, is of special importance in the generation of 3D computer images. Many such images are built up from triangle meshes, and as specialized hardware render and shade triangles at low latencies, the performance bottleneck is actually in feeding the triangular structure into the hardware. A significant speedup can be achieved by not feeding every triangle by itself, but by combining them into triangle strips. An adjacent triangle can be defined by only one new point from the previously fed triangle, and therefore adjacent triangles combined in a single strip can speedup the feeding procedure by a maximum factor three for each 3D object. Finding a single strip that incorporates all triangles in the mesh is equivalent to finding a Hamiltonian cycle through the corresponding cubic graph in which every triangle is a vertex, which makes both Eppstein's and Iwama&Nakashima's result of crucial importance for the 3D imagery business (see Figure 2).

So, concludingly, none of the complete algorithms on finding Hamiltonian cycles runs faster than exponential on all instances (with vertices of any degree), and the Bellman–Held–Karp ”is still the strongest known”, as Andreas Björklund states on the first page of his 2010-paper [22]. Cetal's algorithm however, is much closer related to the depth-first approach by Rubin and Martello and runs in $O(n!)$ time, as do the other algorithms in the next section.

III. Three Hamiltonian algorithms

Naked depth-first-search is a complete and exhaustive method, but Cetal mount the algorithm with ”two heuristics”: 1) the starting vertex is the vertex with the highest degree and 2) when recursing, always prioritize a higher degree vertex over a lower degree vertex. It should be very clearly understood though, that Cetal's ”heuristics” are just speedup procedures expected to boost the algorithm's performance by reducing runtimes but do not compromise its completeness like a 'heuristic algorithm' such as Simulated Annealing would.

For this replication, two more algorithms were implemented: Van Horn's algorithm (named after the first author) which is identical to Cetal's, except that it *inverts* the heuristic, starting at the lowest degree vertex, and prioritizing lower degree adjacent vertices over higher ones when recursing. Inspiration for this inversion came from an almost prophetic statement by James Bitner and Edward Reingold from their famous '75-paper that ”In general, nodes of low degree should occur early in the search tree, and nodes of high degree should occur later.” [24]. They were guessing at the time, but it turns out they were right and their intuition has later been formalized in a more generalized way [25], but still it is baffling to see how much performance is gained from such a simple inversion – at zero extra cost.

Thirdly, we replicated Vacul's somewhat more sophisticated algorithm intentionally programming it from scratch for the sake of replicability. It is also a depth-first-search and it also prioritizes vertices of lower degree over higher degree but additionally, it has an iterated-restart feature and three pruning

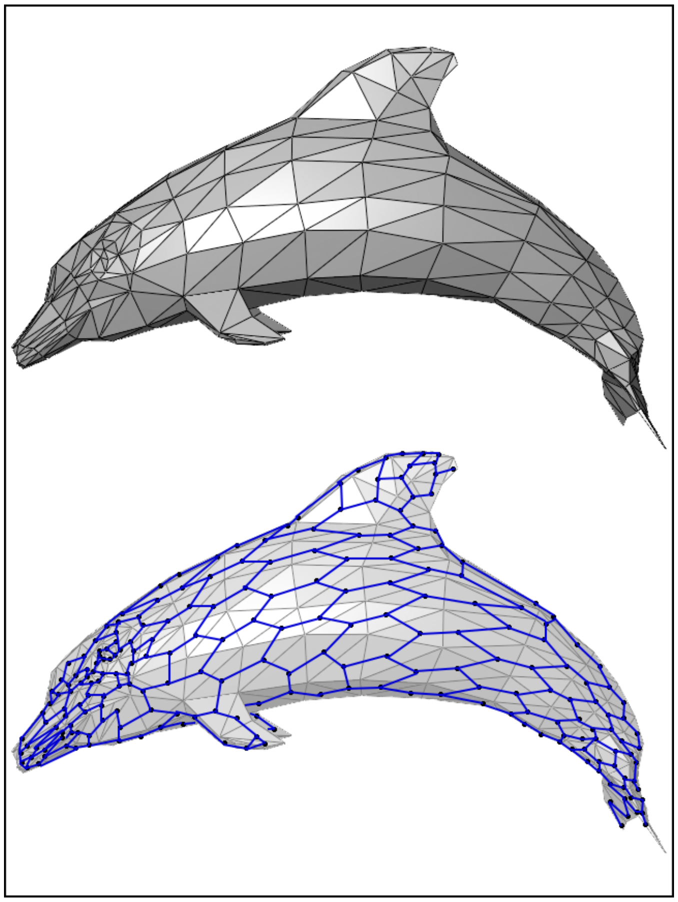

Fig. 2. Fast rendering of triangle mesh 3D images critically depends on finding Hamiltonian cycles through the corresponding 'cubic' graphs, in which every vertex has a maximum degree of three.

techniques, employed in preprocessing and during recursion. A preprocessing stage first runs an iterated pruning routine that chips away edges that cannot be in any Hamiltonian cycle. Any vertex v that has two neighbours (adjacent vertices) n_1 and n_2 which both have degree 2 only keeps those edges to n_1 and n_2; all its other edges are pruned off because they cannot be in any Hamiltonian cycle. It also looks for a forced path, and prunes the edge between the first and last vertex if it exists. Since any pruning activity can lead to new vertices of degree 2, the procedure needs to be run again until no more edges are pruned off. Then, the graph is checked for non-Hamiltonicity-properties: degrees smaller than 2 and articulation points (a.k.a "cut vertices"). If it has neither of those two, the preprocessing stage is finished and the depth-first recursion starts. During recursion, the exact same pruning methods and one more take place: whenever a vertex is added to the path, all edges from the previous node are pruned, except for the two in the path. As is common for depth-first, all pruned edges get placed back when backtracking. Finally, Vacul set an upper bound on the number of recursions for their algorithm. When exceeded, the bound is upped and a random restart is launched. An interesting technique, but we omitted it simply because Cetal's original graph sizes are small enough to do an exhaustive search – Vacul's go up to 1500 vertices, Cetal's only to 24. A peculiar detail to keep in mind is that these kinds of small randomnesses often improve runtimes in exact algorithms on large inputs, but also make them non-deterministic: experiments may vary from trial to trial - even for two runs on the same input. By omitting this option, runtimes (such as in Figure 3) are consistent, exactly reproducible and non-variable on their input.

IV. Experiment & Results

In Cetal's study, 20 graphs were generated for "a given connectivity", and analyzed for Hamiltonian Cycles and the computation times (in iterations) recorded. Some runs are cut off at "a prespecified maximum". The average is taken, but it "[severely underestimates the true costs, as it also contains those saturated values]". So, although of the exact experimental parameters are left undefined, Cetal state: "[The existence of the phase transition is clear]".

This left us with some choices. Generally, we tried to stick to Cetal's work as closely as possible, generating two sets of graphs, one with 16 vertices and one with 24 vertices and employed the full range of 0 to the maximum $\frac{1}{2}n^2 - \frac{1}{2}n$ edges. We generated 20 random graphs for every number of edges, resulting in 2400 random graphs for the 16-vertex graphs, and 5520 random graphs for the 24-vertex graphs. We could not consult Cetal's source data, but judging by their figures, our numbers might be higher than Cetal's original work, which makes our replication a little more rigourous. In the remainder of the paper, we will discuss results on the 24-vertex graphs, but results for the 16-vertex graphs are comparable and the reader is encouraged to try our online interactive diagrams when interested.

We ran Cetal's algorithm, Van Horn's algorithm, and Vacul's algorithm all on the same input data and recorded the number of iterations for each algorithm on each random graph. We set our cutoff point at 10^9 iterations, which was reached 65 times by Cetal's algorithm and 57 times by Van Horn's algorithm (but not on the same graphs). Van Horn's algorithm, with its inverted heuristic, performs better with shorter runtimes on 4627 out of the 5520 graphs (83.8%), but nonetheless peaks in runtime near the Hamiltonian phase transition. Vacul's algorithm with advanced pruning outperformed both other algorithms on 3334 out of the 5520 graphs (61.3%). It did not reach the cutoff point even once, even though its hardest graphs are still near the Hamiltonian phase transition, roughly between vertex degrees 4 and 7 for 24 vertices. Note that the vertical axis is logarithmic, so the computational difference between easy and hard graphs is astoundingly large, even for these small instances.

V. Conclusion and Discussion

It seems fair to say that Cetal's results on the Hamiltonian cycle problem are reproducible and valid. Harder graphs do reside around the Hamiltonian phase transition for their algorithm and the average vertex degree indeed functions as

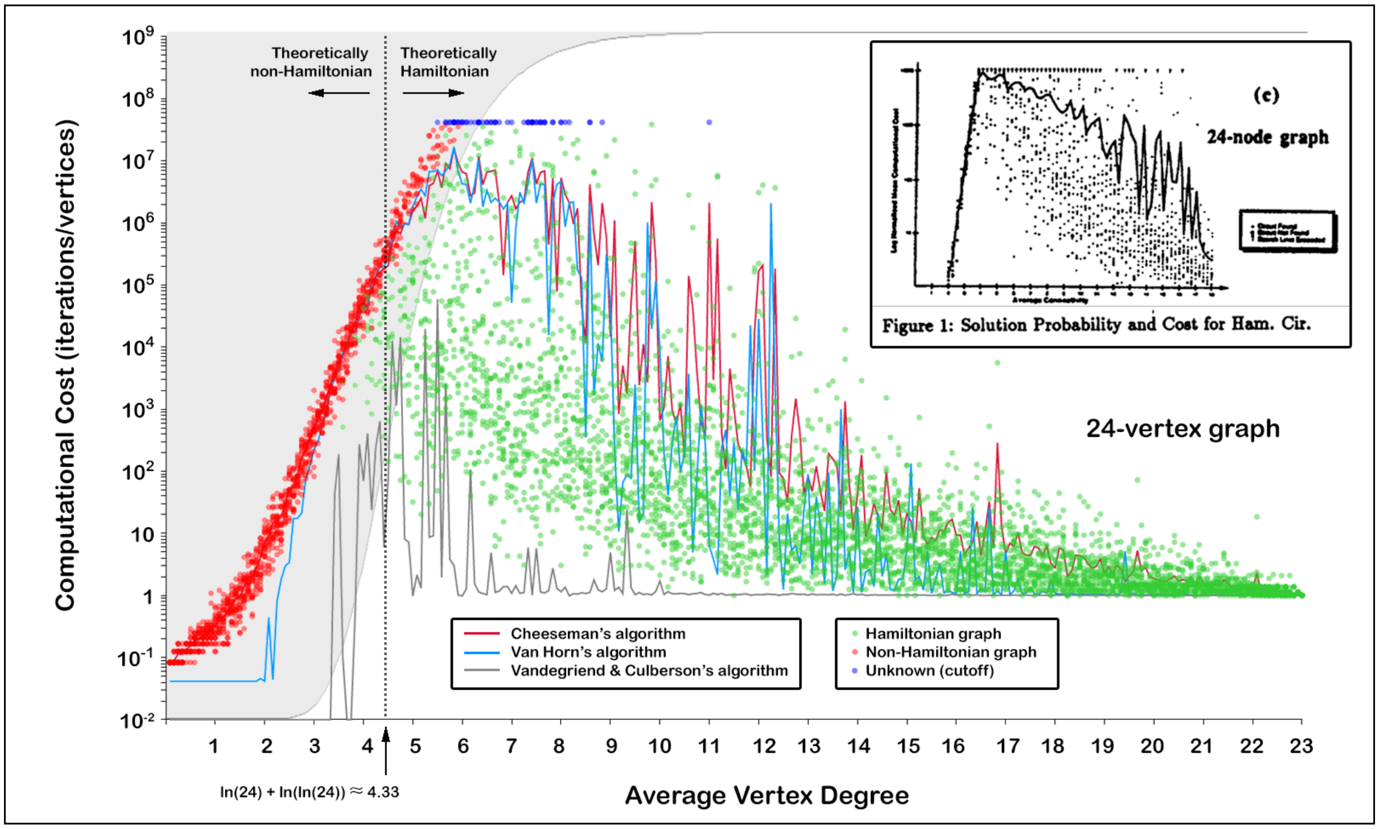

Fig. 3. Results of a replicated study by Cetal, which we extended with algorithms by Van Horn and Vacul. The top-right inset is Cetals original figure, and it covers no data points. Note how the order parameter is a useful predictive data analytic for all three algorithms.

an order parameter, and is to some degree useful for predicting runtimes. Furthermore, the same order parameter also seems to hold for Van Horn's algorithm, which largely outperforms Cetal's with shorter runtimes simply by inverting the heuristic from prioritizing higher degree vertices to prioritizing lower degree vertices.

Finally, Vacul's algorithm of sophisticated pruning lives up to its promise and outperforms both other algorithms, being so effective it practically erases the phase transition for our problem instances. Still, the harder graphs are near the phase transition and the average vertex degree might still be seen as an order parameter. Much of this is due to the pruning done during preprocessing and recursing. This procedure however, is quadratic-time and gets executed during every iteration in an exponential algorithm. So, even though the number of iterations may be much lower, the amount of wall clock time per iteration is likely to be higher. To what extent this is a purely theoretical issue remains to be seen; the cost-benefit tradeoff of such 'intermediate speedup procedures' is classically related to constructive algorithms, and we would like to quantitatively investigate the details for these specific algorithms in future work.

For now, these pruning procedures appear quite beneficial and firmly instantiate Steven Skiena's famous take-home lesson "[Clever pruning can make short work of surprisingly hard problems.]" [26]. Furthermore, Bitner & Rheingold's early observation that nodes with tighter constraints should be prioritized over looser constraints seems like a good guiding principle for designing these kinds of complete algorithms – at least for Hamiltonian cycle detection, but possibly for a much wider class of problems because after all, it (still) is NP-complete.

With the seminal work of Peter Cheeseman, Bob Kanefsky and William Taylor, the field of instance hardness has seen the light. Ever since, principles such as solution backbones, complexity cores and algorithm selection have all emerged as important scientific concepts, scraping ever more plaster off the wall that separates P from NP. We think these concepts, and their paper, should be part of any serious curriculum in computer science or artificial intelligence.

VI. Acknowledgements

Many thanks to Richard Karp, Edward Rheingold, Andreas Björklund and Joseph Culberson for answering questions over email, and to Marcel Worring and Hans Dekkers from University of Amsterdam for granting some research time.

References

[1] R. E. Bryant, "On the complexity of vlsi implementations and graph representations of boolean functions with application to integer multiplication," IEEE transactions on Computers, vol. 40, no. 2, pp. 205–213, 1991.

[2] F. Ivančić, Z. Yang, M. K. Ganai, A. Gupta, and P. Ashar, “Efficient sat-based bounded model checking for software verification,” Theoretical Computer Science, vol. 404, no. 3, pp. 256–274, 2008.
[3] P. C. Cheeseman, B. Kanefsky, and W. M. Taylor, “Where the really hard problems are.” in IJCAI, vol. 91, 1991, pp. 331–340.
[4] T. Larrabee and Y. Tsuji, Evidence for a satisfiability threshold for random 3CNF formulas. Citeseer, 1992.
[5] S. Kirkpatrick and B. Selman, “Critical behavior in the satisfiability of random boolean expressions,” Science, vol. 264, no. 5163, pp. 1297–1301, 1994.
[6] I. P. Gent and T. Walsh, “Easy problems are sometimes hard,” Artificial Intelligence, vol. 70, no. 1-2, pp. 335–345, 1994.
[7] T. Hogg and C. P. Williams, “The hardest constraint problems: A double phase transition,” Artificial Intelligence, vol. 69, no. 1-2, pp. 359–377, 1994.
[8] T. Hogg, “Refining the phase transition in combinatorial search,” Artificial Intelligence, vol. 81, no. 1-2, pp. 127–154, 1996.
[9] I. P. Gent and T. Walsh, “The tsp phase transition,” Artificial Intelligence, vol. 88, no. 1-2, pp. 349–358, 1996.
[10] M. R. Garey and D. S. Johnson, Computers and intractability. wh freeman New York, 2002, vol. 29.
[11] A. M. Jaffe, “The millennium grand challenge in mathematics,” Notices of the AMS, vol. 53, no. 6, pp. 652–660, 2006.
[12] P. Erdos and A. Rényi, “On the evolution of random graphs,” Publ. Math. Inst. Hung. Acad. Sci, vol. 5, no. 1, pp. 17–60, 1960.
[13] L. Pósa, “Hamiltonian circuits in random graphs,” Discrete Mathematics, vol. 14, no. 4, pp. 359–364, 1976.
[14] J. Komlós and E. Szemerédi, “Limit distribution for the existence of Hamiltonian cycles in a random graph,” Discrete Mathematics, vol. 43, no. 1, pp. 55–63, 1983.
[15] S. Roberts and B. Flores, “Systematic generation of Hamiltonian circuits,” Communications of the ACM, vol. 9, no. 9, pp. 690–694, 1966.
[16] M. Held and R. M. Karp, “A dynamic programming approach to sequencing problems,” Journal of the Society for Industrial and Applied Mathematics, vol. 10, no. 1, pp. 196–210, 1962.
[17] R. Bellman, “Dynamic programming treatment of the travelling salesman problem,” Journal of the ACM (JACM), vol. 9, no. 1, pp. 61–63, 1962.
[18] S. Martello, “Algorithm 595: An enumerative algorithm for finding Hamiltonian circuits in a directed graph,” ACM Transactions on Mathematical Software (TOMS), vol. 9, no. 1, pp. 131–138, 1983.
[19] F. Rubin, “A search procedure for Hamilton paths and circuits,” Journal of the ACM (JACM), vol. 21, no. 4, pp. 576–580, 1974.
[20] B. Vandegriend and J. Culberson, “The gn, m phase transition is not hard for the Hamiltonian cycle problem,” Journal of Artificial Intelligence Research, vol. 9, pp. 219–245, 1998.
[21] B. Bollobas, T. I. Fenner, and A. M. Frieze, “An algorithm for finding Hamilton paths and cycles in random graphs,” Combinatorica, vol. 7, no. 4, pp. 327–341, 1987.
[22] A. Bjorklund, “Determinant sums for undirected Hamiltonicity,” SIAM Journal on Computing, vol. 43, no. 1, pp. 280–299, 2014.
[23] K. Iwama and T. Nakashima, “An improved exact algorithm for cubic graph tsp,” in International Computing and Combinatorics Conference. Springer, 2007, pp. 108–117.
[24] J. R. Bitner and E. M. Reingold, “Backtrack programming techniques,” Communications of the ACM, vol. 18, no. 11, pp. 651–656, 1975.
[25] Gent et al., “The constrainedness of search,” in AAAI/IAAI, Vol. 1, 1996, pp. 246–252.
[26] S. S. Skiena, “The algorithm design manual,” p. 247, 1998.