Supplementary Material for: Repulsive Curves

CHRIS YU, Carnegie Mellon University
HENRIK SCHUMACHER, RWTH Aachen University
KEENAN CRANE, Carnegie Mellon University

1 SUMMARY

This supplemental document details the dataset and the methods that were used in the evaluation. The datasets and results themselves can be found in the `KnotDatasets` and `Results` directories, *resp.*

1.1 Dataset

In order to generate random embeddings of knots with fixed isotopy classes, we used *KnotPlot* [Scharein 1998] to randomly perturb knots in a nonintersecting manner, starting from the canonical embeddings provided in *KnotPlot*'s "knot zoo." A strong thermal energy combined with a weak elastic energy was used to randomize the initial configurations; *KnotPlot* prevents collisions, preserving the initial isotopy class. The exact *KnotPlot* script that was used can be seen in Figure 1.

Initially, we created 128 randomly embedded knots of distinct knot classes, corresponding to the first 128 entries of *KnotPlot*'s knot zoo. We then additionally sampled 100 distinct trefoil knots, for a set of 228 randomly embedded knots total. All of these knots were intentionally sampled to be difficult in terms of geometric complexity; this was achieved by setting a minimum threshold on the file size (which is almost directly proportional to the vertex count) of the curves exported by *KnotPlot* and resampling all generated knots until they exceeded this threshold. The threshold chosen was 60 KB, which corresponds to about 1500 vertices.

1.2 Experiments

We implemented a set of 19 different optimization methods (including ours) to minimize the tangent-point energy $\mathcal{E}^2_{4.5}$ in Mathematica; specific details about the set of methods are deferred to Section 2. All experiments were run single-threaded on an Intel Xeon E5-2690 processor with 196 GB of RAM. All derivatives were computed exactly using symbolic differentiation; no numerical approximations were used. We tested the full battery of 19 methods on examples at multiple resolutions. We used UMFPACK for sparse linear solves and LAPACK for dense linear solves. We selected the Freedman unknot, a random knot of a nontrivial knot class (numbered 9_6), and a random trefoil knot, and generated samplings of these knots ranging from about 250 up to about 4000 vertices via subdivision. Graphs of the results of these comparisons are shown in the article, and the data are also available in other supplemental materials.

```
knot number 10
cyl-rad 0.0001
hooke = 0.1
force 1
therm = 1
thfstr = 1
stusplit = 2
ago 200
force 2
therm = 0
hooke = 1
charge = 20
ago 50
```

Fig. 1. A *KnotPlot* script for randomly generating knot embeddings.

To save on runtime for the full dataset, we cut several poorly performing methods, leaving only the most competitive methods to be run on all 228 knots. The results of this experiment have been aggregated into scatter plots in the article, and the complete data have also been included in the supplemental materials. We found a small number of knots (less than 20) were so complex that *none* of the methods we tested were able to make progress without becoming stuck due to collisions; these examples provided no useful comparisons, so we did not include them in our scatter plots, but their data is also in the supplemental materials to leave the door open for comparisons against future work.

1.3 Stopping Criteria

For all experiments in this dataset, all optimization methods continued to run until either of the following two conditions was met:

(1) The L^2 norm of the preconditioned gradient is below a fixed tolerance $\epsilon = 10^{-4}$, or
(2) A fixed amount of wall-clock time has elapsed since the method began running.

Condition (1) is identical to the one described in the main body of the article. Condition (2) was imposed simply because some methods exhibit such poor convergence that they would require days to satisfy the first condition. The time limit for condition (2) was 24 minutes per example for everything except the finest multiresolution examples, for which it was 2 hours.

2 METHODS OVERVIEW

We implemented 19 different optimization methods to examine the effectiveness of our proposed H^s strategy. These methods cover

© 2021 Association for Computing Machinery.
0730-0301/2021/05-ART10 $15.00
https://doi.org/10.1145/3439429

a broad range of optimization techniques, including second-order methods and quasi-Newton methods using a variety of preconditioning strategies. We group these methods into categories based on the high-level approach; while differences between categories are significant, methods within each category vary by relatively small implementation changes such as choice of inner product.

Unless otherwise indicated, all gradient flows are integrated using explicit (forward Euler) timestepping with line search; the specific line search strategy is described in the main article.

2.1 Choices of Preconditioner

Methods within a category are mostly differentiated by the choice of inner product. While the application of the inner product differs between categories, within each category, the inner product is essentially used the same way, and different inner products (represented by different matrices A) can more or less serve as drop-in replacements for each other. The inner products we consider are:

- L^2 – the Euclidean inner product, corresponding to the mass matrix M of the curve.
- H^1 – the stiffness matrix L of the weak Laplacian.
- H^2 – the curve Laplacian squared, i.e., $LM^{-1}L$.
- H^s – our fractional Sobolev-Slobodeckij inner product, assembled using the technique described in the article.
- Convexified Hessian – the sum of all positive semidefinite projections of local Hessians from all edge-edge pairs. Gradient descent using this preconditioner is also sometimes called "projected Newton's method," but we do not use that term here to avoid confusion with constraint projection methods. See Section 3.6 for more details.

2.2 Hard Constraints and Soft Penalties

As mentioned in the article, some methods enforce hard constraints on, e.g., edge lengths, while other methods are not designed to do so, and instead enforce these constraints using soft penalty terms. The projected gradient method described in the article falls in the former category, while the latter category includes well-known methods like L-BFGS and nonlinear conjugate gradients.

The performance of methods enforcing hard constraints cannot be compared directly to those using soft constraints, as minimizers reached using soft constraints are generally not feasible under hard constraints, and energy values also differ due to the presence or absence of penalty terms. Therefore, in addition to the H^s projected gradient method we describe in the article, we also implemented several H^s-preconditioned soft constraint methods, in order to obtain an apples-to-apples comparison with other soft constraint methods. We plot these two types of methods separately, with H^s projected gradient descent acting as "our" representative method using hard constraints, and H^s nonlinear conjugate gradients acting as "ours" using soft constraints. We note that methods using soft penalty constraints should be expected to perform better than similar methods using hard constraints, since allowing trajectories to leave the constraint manifold can only result in more direct paths to minimizers than requiring trajectories to remain on the manifold.

Throughout our comparison, for all methods capable of enforcing hard constraints, we used fixed barycenter and fixed edge length constraints. The barycenter constraint factors out global translations that do not affect the energy, while the edge length constraints ensure that a minimizer exists. Without constraining edge lengths, the curve is free to expand infinitely; this allows curves to simply grow infinitely without ever untangling, making the optimization problem ill-defined. For problems that cannot handle hard constraints, we only require the gradient to be orthogonal (in the L^2 sense) to the space of constant vector fields, while replacing the edge length constraint with Hencky strain terms in the objective for each edge, weighted by edge lengths of the initial configuration; this is a standard nonlinear model of elasticity whose linearization corresponds to standard linear elasticity.

3 SPECIFIC METHODS

3.1 Projected Gradient Descent (Hard Constraints)

Projected gradient descent methods follow the exact framework outlined in the main article. Given an energy function $\mathcal{E}$, an inner product A, and a constraint function whose differential is stored in the matrix C, a projected gradient method solves the saddle problem

$$\begin{bmatrix} \mathsf{A} & \mathsf{C}^\mathsf{T} \\ \mathsf{C} & 0 \end{bmatrix} \begin{bmatrix} g \\ \lambda \end{bmatrix} = \begin{bmatrix} d\mathcal{E} \\ 0 \end{bmatrix}$$

to obtain the projected descent direction g. The only difference between methods in the category is the choice of inner product A. We implemented versions of this method under the H^s, H^1, H^2, L^2, and convexified Hessian inner products. For H^s and L^2, we also implemented variants that used implicit timesteps rather than explicit.

3.2 Nonlinear Conjugate Gradients (Soft Constraints)

This category consists of implementations of the method of nonlinear conjugate gradients (NCG); a comprehensive introduction to this method is beyond the scope of this document, but several overviews exist [Shewchuk 1994; Hager and Zhang 2006]. Various heuristics exist for updating the conjugate direction in each timestep; we use that of Polak and Ribiere [1969] for all of our comparisons. We apply standard unconstrained NCG to the penalized version of the tangent-point energy.

NCG methods are generally formulated assuming the L^2 inner product. Using a different inner product A amounts to evaluating all gradients and inner products in the NCG algorithm using the new inner product; this essentially serves as preconditioning for NCG. We implemented versions of NCG using H^s, H^1, H^2, L^2, and convexified Hessian inner products.

3.3 L-BFGS (Soft Constraints)

This category consists of implementations of the limited-memory BFGS method [Liu and Nocedal 1989]. We used a memory size of 30 for all L-BFGS methods we tested.

All L-BFGS methods use the same update rule. The only difference between them lies in the initial approximation of the inverse Hessian. Standard L-BFGS uses a "frozen" initial approximation of the Hessian, taken as the inverse of the inner product of the initial curve. We tested both this standard method and a "dynamic" method where the initial approximation is not static, but is taken to be the inverse of the inner product of the *current* curve. Through testing, we found that the "dynamic" methods performed much better, so we only report these methods in our plots. L-BFGS with H^1 in particular is sometimes called "Sobolev-initialized L-BFGS."

All L-BFGS methods use soft penalties. We implemented versions of L-BFGS using H^s, H^1, and L^2 inner products.

3.4 Nesterov Methods (Soft Constraints)

The methods in this category are implementations of Nesterov's accelerated gradient method [Nesterov 1983]. This is essentially a "heavy ball" momentum method, with the stipulation that the gradient step is taken after the momentum step, rather than the two steps occurring simultaneously. All of these methods also implement "adaptive restarting" [O'Donoghue and Candès 2015], where momentum is reset whenever it causes the objective function to increase. Momentum steps can often be nonmonotonic due to the lack of line search, and this modification aims to prevent such steps. As with nonlinear conjugate gradients, running a Nesterov method under a different inner product A equates to using that inner product to evaluate all gradients and inner products between vectors.

In general, it is difficult to force momentum steps to respect hard constraints while still obtaining any acceleration from them, so our Nesterov methods also use soft penalty constraints. Additionally, momentum steps can easily cause curve edges to "tunnel" through each other, since they ignore self-intersections. We therefore used collision detection to prevent this. We implemented versions of this Nesterov method using H^s, H^1, and L^2 inner products.

3.5 Accelerated Quadratic Proxy (Soft Constraints)

This category implements the method of Kovalsky et al. [2016]. AQP bears many similarities to the Nesterov methods from the previous section; the primary differences are:

(1) the inner product is frozen to be that of the curve on the first timestep, instead of recomputed after every step, and
(2) the Nesterov momentum step size is prescribed as $\frac{1-\sqrt{\kappa^{-1}}}{1+\sqrt{\kappa^{-1}}}$, where κ is the condition number of a matrix that can be derived from the inner product.

In practice, κ is estimated rather than analytically computed. We use $\kappa = 1000$, which was also used in the original AQP implementation. AQP only supports linear constraints, while our constraints are nonlinear, so we implemented our constraints using soft penalty terms. We also impose the same collision detection on the momentum step to prevent "tunnelling," which is otherwise frequent.

We first implemented AQP exactly as described in the original article, which uses the H^1 inner product; this is the version we refer to by "AQP" in our comparisons. We then additionally implemented a version that differs only by using our H^s inner product instead of the H^1 inner product. We call this method "H^s (fractional) AQP" in our comparisons.

3.6 Convexified Hessian

While the two methods using the "convexified Hessian" inner product have already been mentioned in other categories, the inner product is different enough to warrant further exposition. The goal of these two methods is to run Newton's method using the projection of the Hessian onto the positive semidefinite cone. However, performing this projection on the full Hessian is prohibitively expensive. Therefore, the projection is instead done per term. For each energy term, we compute its local Hessian, project it to be positive semidefinite, and then add it to the appropriate indices of a global Hessian. In our case, every pair of distinct edges produces a distinct energy term, so there are $O(|E|^2)$ local Hessians to project.

Assuming that the Hessian H is real-valued, its positive semidefinite projection under the mass matrix M can be found by computing the generalized eigendecomposition $HQ = MQ\Lambda$, clamping all of the negative eigenvalues in Λ to 0, and multiplying the matrices back together again.

The two convexified Hessian methods we implemented were a version of the projected gradient method and a version of nonlinear conjugate gradients; details about what these methods do after the inner product has been computed can be found in the respective sections.

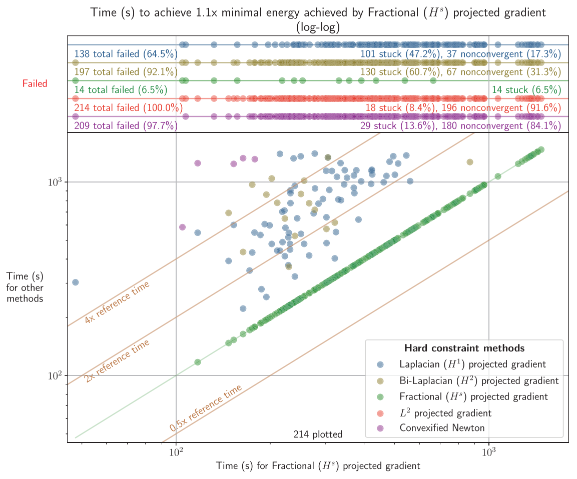
4 RESULTS

We visualized the results of our comparison in several ways. Firstly, for all examples, we directly plotted the objective value attained by each method against both elapsed wall-clock time and iteration count. This resulted in 4 plots per example, showing both wall-clock time and iteration counts for both hard constraints and soft constraints.

Next, to better capture the relative performance of methods on the full dataset of 228 knots, we aggregated the performance for all methods on all examples into 4 log-log scatter plots, again corresponding to wall-clock time and iteration count for hard and soft constraints. Each scatter plot has a chosen "reference" method, and the plotted points indicate the time taken for the reference to reach a certain energy threshold on a given example, versus the time it took for another method to reach the same threshold on the same example. Each example has a separate threshold, which was chosen to be 1.1× the minimal energy reached by the reference method on that example. As a result, the points for the reference method are always on the line $y = x$, and points above this diagonal for competing methods indicate that those methods were slower, while points below the diagonal indicate that those methods were faster.

Along the top of the scatter plots, there is a line for each method showing instances where that method failed to reach the target threshold. This can be for one of two reasons. Firstly, the method may become unable to progress due to its line search step size collapsing to zero, a failure mode that we call "stuck"; this generally happens because the method produces a search direction that would cause an existing near-collision to become a collision, which no amount of backtracking can resolve. Or, secondly, the method may continue to make progress, but still fail to reach the target energy threshold within the time limit (see Section 1.3) that we set for each experiment, a failure mode that we call "nonconvergent."

The reference method cannot be nonconvergent, since the target energy threshold is based on the minimum energy attained by the reference; however, the reference method can still become stuck, which we record. In this case, we still set the target energy based



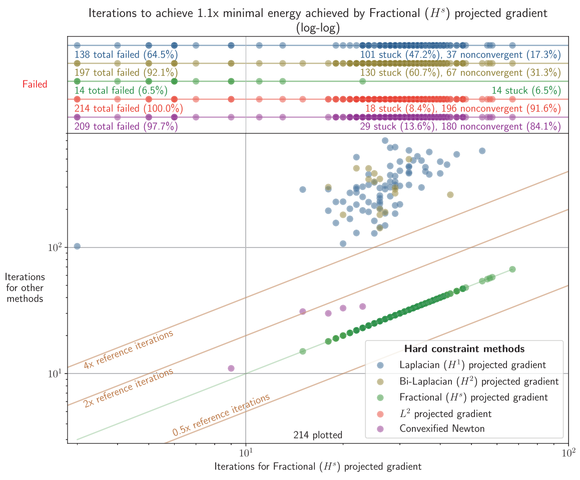


Fig. 2. *Top:* A log-log scatter plot illustrating the wall-clock performance of all hard constraint methods, relative to that of our H^s projected gradient method. Lines are drawn that indicate 0.5×, 2×, and 4× the time of our method. *Bottom:* Same as on top, but for iteration counts.

on the minimum value it attained before getting stuck, which can theoretically be a large "win" for methods that do not get stuck on the same instance. But in practice, when the reference seldom gets stuck, most other methods also get stuck.

Overall, the results show a considerable speedup in both wall-clock time and iteration counts in favor of our fractional H^s methods. We note that, due to the limitations of Mathematica, no spatial acceleration is applied; further, our H^s methods are using dense linear solvers, while other metrics such as H^1 are able to use far more efficient sparse solvers. That our methods are already outperforming others using dense linear algebra shows the power of our fractional operators; adding in the accelerations we describe in the article can only improve our performance further.

4.1 Hard Constraint Methods

Aggregate scatter plots for the hard constraint methods are shown in Figure 2. The chosen reference method was H^s projected gradient descent. The wall-clock plots show that H^s projected gradient descent is considerably faster in real time than all other hard constraint methods we tested; the next-best method was H^1 projected gradient descent, which was generally 2 to 4 times slower on methods where it reached the threshold. Equally if not more notable, however, is the fact that the non-H^s methods are frequently unable to reach the target energy, due to being either stuck or nonconvergent. The best-performing competitor is again H^1, whose failure rate is 64.5% from all causes, including a stuck rate of 47.2%; in contrast, H^s gets stuck on only 6.5% of plotted examples. (These percentages exclude the 14 knots that were so difficult that all hard constraint methods became stuck.) The advantage in iteration counts is even more apparent, with only convexified Newton coming close in this metric on the few examples where it successfully reached the target.

The plots of energy versus time from the multiresolution experiments confirm this advantage and show that the difference is more pronounced with more vertices. Figure 3 shows plots for the same curve at 1024, 2048, and 4096 vertices. H^s outperforms all other methods at all three of these resolutions, but H^1 is quite competitive for the first two. At the highest resolution, however, H^1 worsens significantly, leaving H^s as the clear leader. Similar patterns hold for the other multiresolution experiments, with other methods suffering far more from increased resolution than H^s.

This reflects our discussion of matching the inner product to the energy; our H^s inner product matches the order of the energy and therefore does not suffer limitations from increased resolution, unlike other inner products such as H^1. This is corroborated by plots of iteration counts on the same examples (Figure 4), which show that H^s takes nearly the same number of iterations at all resolutions, while other methods require many more iterations as the vertex count increases. The exception is convexified Newton, since the Hessian also matches the order of the derivatives by definition; however, the PSD projection of n^2 local Hessians is so prohibitively expensive that the method is not competitive in wall-clock time.

4.2 Soft Constraint Methods

The same general trends observed in the hard constraint methods also hold true for soft constraint methods (Figure 5). Among soft constraint methods, H^s NCG and H^s L-BFGS both use our fractional inner product. We used H^s NCG as the reference, since both perform roughly equally well, but H^s L-BFGS gets stuck more often: 14.8% of the time, compared to 3.3% for H^s NCG. (Note that these percentages exclude the 18 examples where all soft constraint methods became stuck.) In contrast, H^1 NCG and H^1 L-BFGS (i.e., "Sobolev-initialized L-BFGS") both take about twice as long to reach the same thresholds on instances where they succeed, but also get stuck over 50% of the time. Meanwhile, AQP [Kovalsky et al. 2016] fares even worse, getting stuck 85.7% of the time, and often taking nearly 4 times as long when it succeeds. Once again, these advantages only grow wider when iteration counts are considered instead of wall-clock time.

Again, the plots of objective values over time for multiresolution experiments are consistent with these observations; the

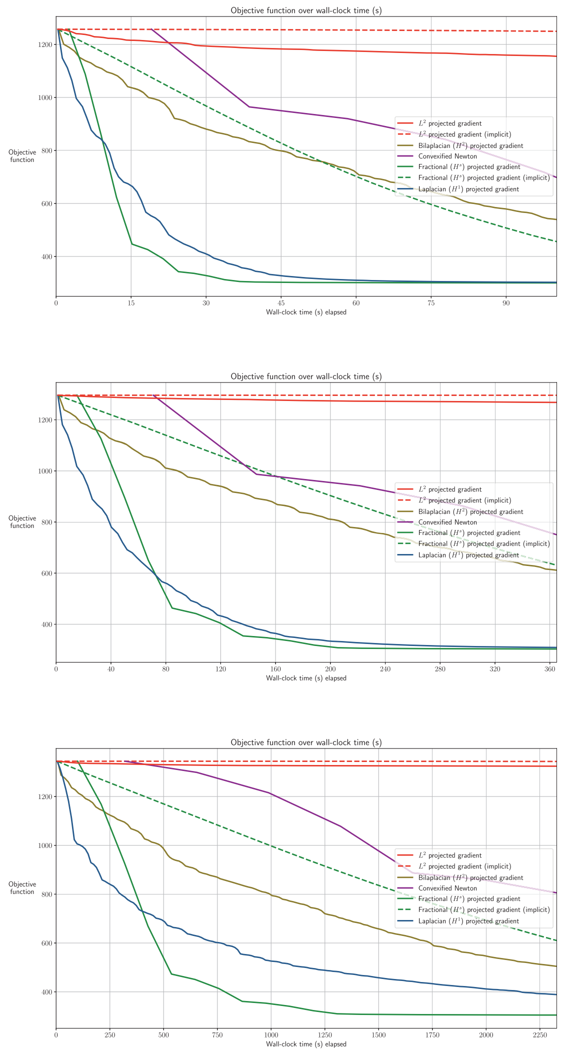


Fig. 3. A plot of objective value over time for all hard constraint methods on one of the multiresolution examples: the knot 9_6 at 1024, 2048, and 4096 vertices (from top to bottom). Note the different x-axis bounds per plot.

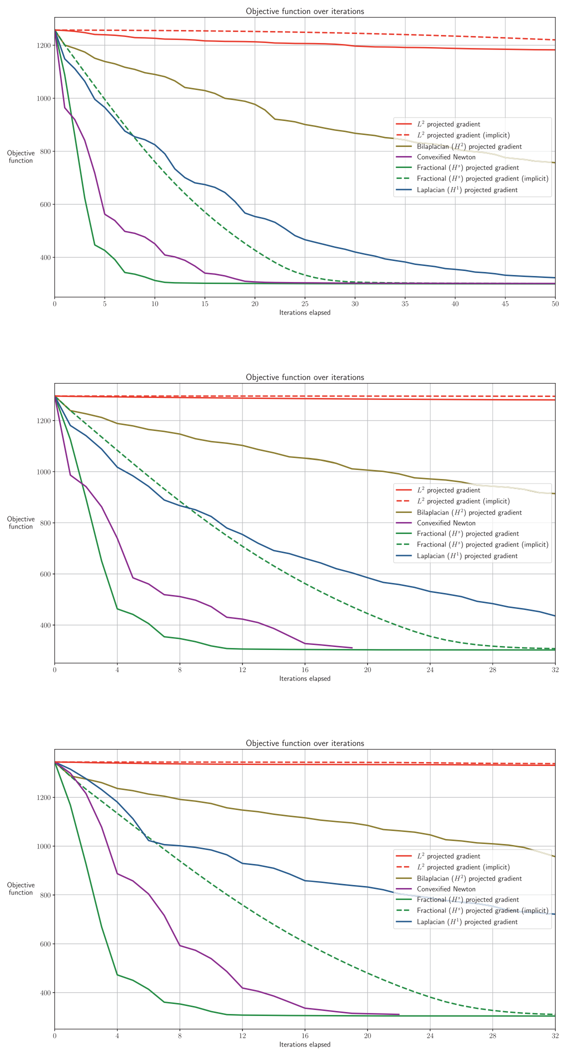


Fig. 4. An analogous set of plots to Figure 3, but showing iteration counts instead of wall-clock time. Note the different x-axis bounds per plot.

H^s methods are generally faster, and this advantage persists or even grows with increased resolution (Figure 6). On examples such as this one where it succeeds, H^1 L-BFGS does a somewhat better job of keeping up with the H^s methods, but it is still about twice as slow to reach a comparable energy value at the bottom.

4.3 Discussion

Our fractional H^s inner product provides a significant speedup over other methods, whether used with hard constraints or soft penalties. Notably, we achieve this speedup despite using dense linear algebra for the H^s methods, versus the faster sparse routines used by other methods. Beyond raw speed, our fractional method is more robust to geometrically complex examples: while other methods either become stuck entirely due to collisions or fail to make satisfactory progress within a reasonable time, our methods rarely become stuck—and only on extremely difficult examples where all or nearly all other methods also fail. Of all the methods we tested, the H^s method thus appears to be best suited to the problem of minimizing tangent-point energy.

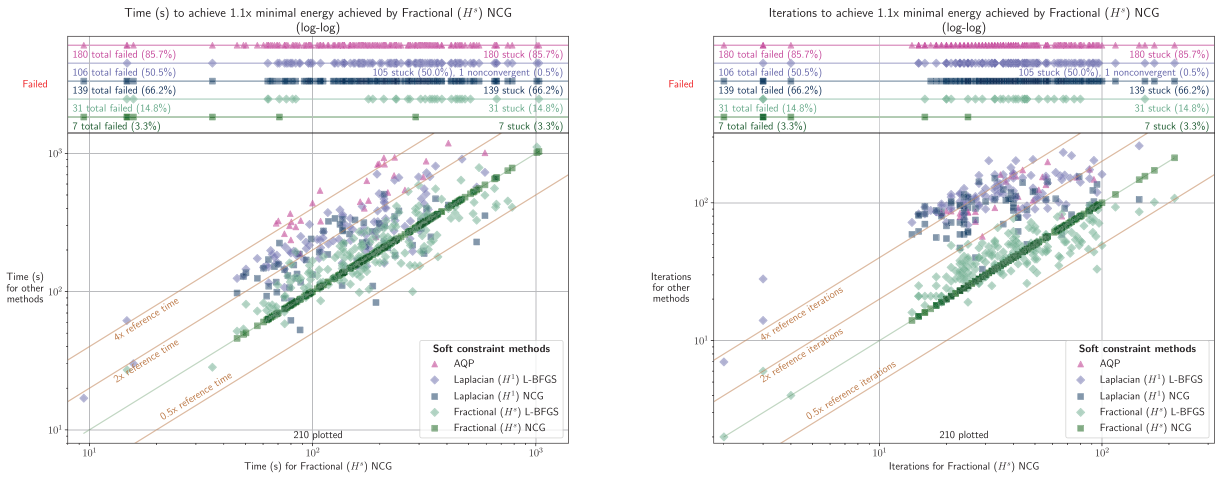


Fig. 5. *Top:* A log-log scatter plot illustrating the wall-clock performance of all soft constraint methods, relative to that of H^s NCG. Lines are drawn that indicate 0.5×, 2×, and 4× the time of our method. *Bottom:* Same as on top, but for iteration counts.

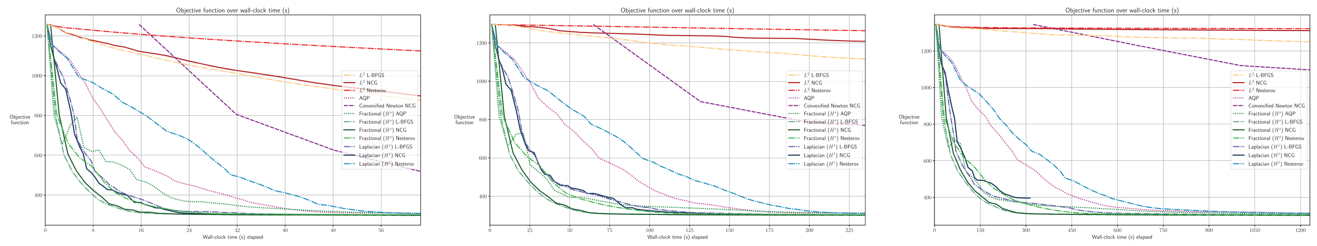


Fig. 6. A plot of objective value over time for all soft constraint methods on one of the multiresolution examples: the knot 9_6 at 1024, 2048, and 4096 vertices (from top to bottom). Note the different x-axis bounds per plot.

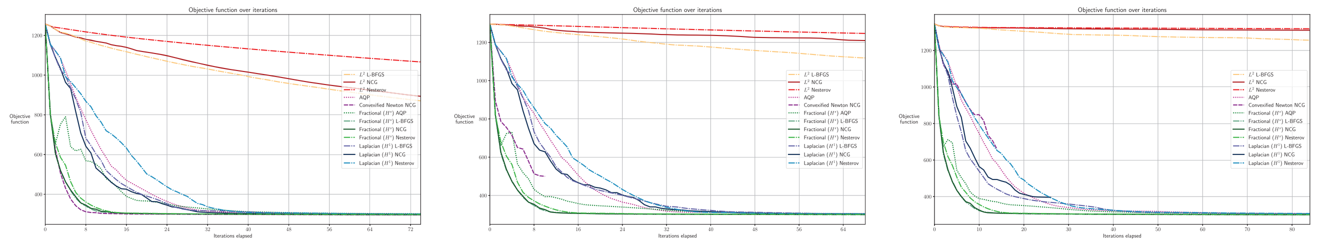


Fig. 7. An analogous set of plots to Figure 6, but showing iteration counts instead of wall-clock time. Note the different x-axis bounds per plot.

REFERENCES

W. Hager and H. Zhang. 2006. A survey of nonlinear conjugate gradient methods. *Pac. J. Optim.* 2 (2006), 35–58.

S. Kovalsky, M. Galun, and Y. Lipman. 2016. Accelerated quadratic proxy for geometric optimization. *ACM Trans. Graph.* 35, 4 (July 2016), 1–11.

Dong C. Liu and Jorge Nocedal. 1989. On the limited memory BFGS method for large scale optimization. *Math. Program.* 45, 1–3 (1989), 503–528.

Y. Nesterov. 1983. A method for solving the convex programming problem with convergence rate $O(1/k^2)$. *Dokl. Akad. Nauk SSSR* 269, 3 (1983), 543–547.

B. O'Donoghue and E. Candès. 2015. Adaptive restart for accelerated gradient schemes. *Found. Comput. Math.* 15, 3 (2015), 715–732.

E. Polak and G. Ribiere. 1969. Note sur la convergence de méthodes de directions conjuguées. *ESAIM: Math. Model. Num. Anal.* 3, R1 (1969), 35–43.

Robert Glenn Scharein. 1998. *Interactive Topological Drawing.* Ph.D. Dissertation. University of British Columbia.

Jonathan Richard Shewchuk. 1994. *An Introduction to the Conjugate Gradient Method Without the Agonizing Pain.* Carnegie-Mellon University, Department of Computer Science.