Fast and Trustworthy Minimum Cost Flows

Mohammad Abdulaziz and Thomas Ammer, King's College London

Abstract

We present a *fast and trustworthy* solver for minimum cost flows utilising certificate checking, formal verification of programs and heuristics in common minimum cost flow algorithms. We first use a theorem prover to formally verify the mathematics and algorithms behind minimum cost flows to produce a scaffolding for calling certifying oracles followed by certificate checking and a verified fallback, if applicable. To obtain oracles, we extend some algorithms from LEMON with the generation of certificates that assert the optimality of a returned flow or infeasibility of the problem. We add some heuristics to the oracles to improve caching and exploit the characteristics of specific instance families. With this approach of implementing mathematical software, we obtain a flexible and maintainable solver with correctness guarantee. The guarantee's overhead decreases at scale, often getting close to zero.

1 Introduction

Speed and correctness of mathematical software, such as the algorithms implemented for the DIMACS implementation challenges, are often seen as a trade-off. Consider Push-Relabel [25]: it is simple from a high-level perspective, yet an efficient implementation of it needs many low-level optimisations, e.g. selection heuristics and relabelling strategies, all of which make it hard to ensure correctness. In addition to complex, low-level optimisations, correctness of sophisticated solvers is hard to maintain during the years of development that such software usually undergoes, since, with growing complexity, tests will not catch every hidden bug.

Formal verification with interactive theorem provers (ITPs) is nowadays an established way of ensuring correctness of algorithms and their implementations by means of a mathematical proof. ITPs are computer programs in which axioms and a conjecture are stated in a formal language; a user provides the ITP with relatively high-level steps of a formal proof for the conjecture; then, by invoking automated theorem provers, the ITP tries to generate low-level steps that fill the gaps between the steps provided by the user, thus forming a complete formal proof of the conjecture. The ITP also checks the entire formal proof against the axioms, thus confirming the soundness of all derivations in the formal proof. In Isabelle/HOL [45], which we use here, that check is performed by a small kernel implementing the inference rules of the logic [43], so bugs in the large set of tools built around the kernel cannot lead to invalid proofs, but only to the kernel refusing a proof. Also, the functional fragment of the logic can be extracted to executable code, e.g. in Standard ML or Haskell, whose correctness follows from the formal proof. Indeed, ITPs have been used to build software whose correctness is mathematically proven in a machine-checkable way, aka formally verified, e.g. an operating system kernel [31], a C compiler [39], and implementations of a multitude of algorithms [44].

In this work, we use an ITP to implement an efficient solver for the minimum-cost problem with correctness guarantees. To build such a solver, there are two methodological options, which were explored in the literature. First, is to verify the solver in full, including all low-level optimisation (e.g. renumbering vertices to improve cache friendliness), which guarantees that its output is correct for every input [38, 3] and that it would not crash, unless it runs out of memory. However,

the effort this takes is high, to the degree it would be infeasible for a state-of-the-art min-cost flow solver, even if AI assistance now reduces it. Also, a fully verified solver is inflexible, since every change to a heuristic or to a data structure requires redoing parts of the proof, and the code exported from a theorem prover is usually slower than a carefully tuned implementation in C++. The second alternative is certificate checking, where a fast unverified solver returns, in addition to its solution, a certificate of that solution's correctness, and only the checker for those certificates is formally verified [40, 8]. Since a checker is much simpler than the solver whose output it checks, verifying it is significantly less work. However, the guarantee obtained this way is only partial: if the check fails/crashes, the program has no correct result to return, i.e. correctness is guaranteed only for those inputs on which the solver happens to produce a valid certificate. Our approach is a hybrid of the two: an unverified solver (henceforth, an *oracle*) returns a solution together with a certificate, a verified checker checks that certificate, and, if the check fails, a verified fallback solver computes a correct result itself, starting from the unverified solver's output. The running time of certificate generation in the oracles and the verified certificate checking show that the overhead of our solver compared to an unverified efficient solver is practically limited to less than twice as slow, and often, below 10%. This validates our approach, as the standard slowdown for fully verified algorithms is usually an order of magnitude.

The input file, in the format of the DIMACS challenge [1], is parsed, and a verified program then reduces the problem to the lower bound normalised form, i.e. the form in which all lower capacities are zero, on which our earlier Isabelle/HOL development of minimum cost flows is based [3]. The reduced problem is passed to one of four oracles, which we derived from LEMON [16]: the network simplex, cost scaling, capacity scaling and successive shortest paths. The oracle returns either a flow together with duals that certify its optimality, or a Gale-Hoffman cut that certifies that the problem is infeasible. A verified checker then checks that certificate, and only if the check fails does the network simplex as verified fallback run, which restores the invariants that a verified solver needs, e.g. strong feasibility of the spanning tree solution, and computes the flow itself. The flow is finally translated back to the original input problem, again by verified code. The verified parts are exported from Isabelle/HOL as imperative Standard ML, while the oracles are written in C++.

2 Background and Methodology

Minimum Cost Flows. We work with directed multigraphs, i.e. directed graphs allowing for parallel edges between vertices. For a vertex x, we call $\delta^+(x)$ the leaving and $\delta^-(x)$ the entering edges. A *flow network* consists of a directed graph over edges $\mathcal{E}$ and vertices $\mathcal{V}$, a capacity function $u : \mathcal{E} \to \mathbb{Z}$, and a lower capacity function $l : \mathcal{E} \to \mathbb{Z}$. The goal is to find a function, i.e. a *flow* $f : \mathcal{E} \to \mathbb{Z}$ satisfying $l(e) \leq f(e) \leq u(e)$ for any edge e. The *excess* of a flow f at the vertex v, $\mathrm{ex}_f(v)$, is the difference between incoming and outgoing flow $\mathrm{ex}_f(v) \stackrel{\text{def}}{=} \sum_{e \in \delta^-(v)} f(e) - \sum_{e \in \delta^+(v)} f(e)$.

We introduce *balances* $b : \mathcal{V} \to \mathbb{R}$ denoting the amount of flow that should be caught or emitted at every vertex. In addition, there is a function $c : \mathcal{E} \to \mathbb{R}$ telling us about the *costs* of sending one unit of flow through an edge. A flow f's *total costs* $c(f)$ are $c(f) = \sum_{e \in \mathcal{E}} f(e) \cdot c(e)$. The set of *feasible flows* is $\{f \mid \forall v \in \mathcal{V}.\ -ex_f(v) = b(v) \wedge \forall e \in \mathcal{E}.\, l(e) \leq f(e) \leq u(e)\}$. We aim to find a *minimum cost flow (MCF)* which is a feasible flow of least total costs [21, 46].

The equivalent problem where $l(e) = 0$ for all edges e, to which we can reduce by subtracting $l(e)$ from $u(e)$ for all e and changing $b(x)$ by $\sum_{e \in \delta^-(x)} l(e) - \sum_{e \in \delta^+(x)} l(e)$ for all vertices x, plays a central role in this extended abstract. We call this the *lower bound normalised form (LBNF)*.

Algorithms include cycle cancelling [32], successive shortest paths (SSP) [29, 11], its refinement by capacity scaling (CAS) [20], cost scaling (COS) [26], and the network simplex (NS) [14].

For example in C++, these algorithms are supported by LEMON [16], while LEDA [42] and Boost [50] offer a subset, and Goldberg gave a standalone implementation of COS [23] ('CS2'). Beyond the bare algorithms, practical efficiency relies on smart heuristics, both evaluated by Kovács [35] employing LEMON. We assume the reader to be somewhat familiar with MCFs and the algorithms; this subsection is only meant as a refresher.

Theorem Proving. Modern-day *theorem provers* like Isabelle/HOL [45], Lean [15] or Rocq [10] can be used to implement algorithms but at the same time they support machine-checked mathematical proofs of the implementation's correctness ('verification'). We already implemented and verified theory and algorithms for MCFs in LBNF, like Orlin's advanced capacity scaling algorithm [47], in Isabelle/HOL [3]. To ease the verification, we use a refinement approach [52] where we formulate algorithms with abstract data types [28] to abstract from implementation details. We already applied this methodology a couple of times, e.g. for graph algorithms like depth-first search [2], for minimum cost flows [3], and for matroids and greedoids [4]. After instantiating with concrete data structures, we can export verified code into functional languages like SML or Haskell. Performance is adversely affected by the absence of $\mathcal{O}(1)$ array access. We mitigate this with the Isabelle Refinement Framework [36] to verify an imperative program in Isabelle/HOL which can be exported to imperative SML with arrays.

Certificate Checking. An alternative (and historically, a predecessor) to full program verification is *certificate checking* where a program not only returns a solution but also information that proves the solution's correctness or optimality, the so-called *certificate* [40]. The certificate can be checked quickly (often linear time) by a simple program. For graph algorithms, this was implemented by Mehlhorn and Näher in the LEDA library [41, 42]. In the context of MCF in LBNF, an optimality certificate for a feasible flow f is a function π on the vertices ('duals'/'potentials') such that for every edge e going from a vertex x to a vertex y (a) if $f(e) = 0 < u(e)$, then $c(e) + \pi(x) - \pi(y) \geq 0$, (b) if $u(e) > f(e) > 0$, then $c(e) + \pi(x) - \pi(y) = 0$, and (c) if $u(e) = f(e) > 0$, then $c(e) + \pi(x) - \pi(y) \leq 0$. These are the *complementary slackness conditions (CSl)*. For infeasibility of an MCF problem in LBNF, the so-called Gale-Hoffman cut can be used: if there is a set of vertices X such that $\sum_{x \in X} b(x) \notin [-\sum_{e \in \Delta^-(X)} u(e); \sum_{e \in \Delta^+(X)} u(e)]$, the MCF problem is infeasible, where we denote the set of edges entering (leaving) X by $\Delta^-(X)$ ($\Delta^+(X)$). The LEDA library returns certificates alongside the solutions to graph problems.

Due to their simplicity, the verification of certificate checkers is much easier than the verification of a full solver. This includes e.g. checkers for some graph problems [8], error bounds in numerics [9], policy iteration [49] and unsolvability of temporal planning [51].

3 Implementation

Architecture. Our main problem is a verified programs' low empirical speed where a slowdown within one order of magnitude e.g. by factor 9 is good. We keep the amount of verified code that is executed during a run of our solver low. The simplest approach is to use an unverified program to initialise the verified one so that the latter already starts close to the optimum. One can use the same algorithm for both phases; even the initialisation might already solve the problem only letting the verified program ensure that there is nothing left to do. An example are Markov Decision Processes (MDPs) [48] where the unverified solver's result can be passed on to the verified one without intermediate processing. Since, unlike MDPs, MCF algorithms have complex invariants that an unverified solver's result might not satisfy (e.g. absence of residual cycles with negative costs for SSP or strong feasibility of the spanning tree solution in case of NS), a verified MCF program cannot be warm-started immediately. To ensure/restore the invariants, all intermediate processing would have to be verified which adversely affects the performance of the whole pipeline and should therefore be avoided. We call this strategy *verified cleanup*.

Competitive implementations of flow algorithms often make use of duals which can naturally also be used as certificates, e.g. in LEDA [40, 42]. Since the certificate check is one linear scan over the flow values, costs and duals, it is naturally fast in comparison to the actual flow computation, and probably a verified checker would add little relative overhead. A certificate checker employing duals is already implemented for policy iteration [49].

For both verified cleanup and certificate checking, one can obtain a verified program to which the unverified solver is an argument. For our MCF solver, we need (a) totality which would be obtained by expensive intermediate processing but also (b) little verification-induced slowdown which a fast (although incomplete) safeguard by certificate checking provides. Because a certificate check usually succeeds, we first give an unverified solver's solution and certificate to a verified checker followed by a verified cleanup in case of failure.

We reuse earlier theory on flows in Isabelle/HOL [3] to verify checker and cleanup. As the earlier developments are based on the LBNF, we (1) first have a verified program to reduce an MCF problem to its LBNF, followed by (2) a collection of unverified certificate-supplying solvers, the so-called *oracles*, (3) a verified checker and (4) a verified cleanup. Step (1) uses the strategy of *verified encoding* that was already used to reliably solve temporal planning with a trustworthy model checker [51]. The sandwich architecture of our correct-by-construction MCF solver is to our knowledge the first time where verified encoding, checking and cleanup are combined. Effectively, it turns every bug affecting correctness into a bug affecting running time. We use oracles and an input file parser implemented in C/C++ and combine them with verified imperative SML code.

Oracles. Kovács' study [35] established LEMON [16] as the state of the art for MCF solvers, which we therefore took as the starting point for our oracles. As a first step we change the NS (as usual, with artificial edges on which we impose a big M penalty to form an initially strongly feasible spanning tree solution), COS, CAS and SSP from LEMON under LEMON's own default configuration to also give certificates. Since the certificate-supplying LEDA often performs significantly worse than LEMON [35] and due to its proprietary nature which hinders tuning, we decided against reusing the former for our pipeline. We also disregard the poorly performing cycle-cancelling methods.

As optimality certificates, we return the duals that are maintained by the algorithms anyway, e.g. by SSP for the Dijkstra searches [18]. In case of COS, these are only approximate duals, where 0 in CSl is replaced with an ϵ (instantiated as $1/n$), which is still enough to certify optimality in integers. To avoid slow floating point and unsound integer divisions, LEMON scales all costs by $\alpha \cdot n$ and continues its main loop until $\epsilon = 1$ where α is the scaling factor. For COS, the certificate is therefore a pair of scaled approximate duals and the factor $\alpha \cdot n$.

The generation of the Gale-Hoffman cut depends on the algorithm: for COS, which searches with Push-Relabel [25, 6] for an initial feasible circulation, we can infer the cut from the labels that partition the vertices into at least two groups by gaps in the appearing labels. For SSP and its refinement CAS, a cut is given by the vertices found reachable in the final unsuccessful Dijkstra search. Only in the case of NS, for which big-M-penalised artificial edges with non-zero flow imply infeasibility, do we run a designated BFS on the residual graph of the graph's actual edges. Note that we take the supply side of the cut as the certificate.

We identify the edges with numbers from 1 up to m. We have arrays (indexed by edge numbers) to consecutively store the edges' first endpoints, second endpoints, capacities and costs. Similarly, the vertices are numbered from 1 up to n and are used to index the balance array.

The default heuristics for all four algorithms, namely NS, COS, CAS and SSP, are those of LEMON to which we add a few more. Some work generally irrespective of the algorithm, like the *vertex* and *edge order reversal*. *Node presolving* removes zero-supply nodes of degrees 0, 1 and 2 and reconstructs the flow and the certifying potential after solving. *Arc presolving* removes zero-capacity edges and merges parallel edges of equal costs. We also support the option to divide

all costs by their greatest common divisor (GCD) to make overflows less likely.

Specifically for the NS, periodic *vertex reordering* renames all vertices by the preorder given by DFS in NS's spanning tree. Like many ordering heuristics, it aims to exploit cache locality effects during iterations over the edges as needed e.g. for the potential update after a pivot which is conducted in the standard threads [30, 22], i.e. maintained pointers to the next vertex in the spanning tree's pre-order. We added other pivot rules, for example the *inward most negative (IMN)* pivot rule [13, 27] adjusted to finite capacities.

To COS, we add, as proposed by Kovács [35], Goldberg's *arc fixing* [23]. We also vary the scaling parameter α and the path length in partial augment-relabel (PAR). The *blocking augment-relabel (BAR)* method works like partial augment-relabel but first drains the current excess vertex before modifying the global update counter and selecting the next vertex. Recall that partial augment relabel pushes flow along paths of fixed length [24]. It finds them by using a DFS and the residual graph in compressed sparse row (CSR) format. We added previously unimplemented rules to select vertices for the push, like selection by approximate maximum excess with a lazy binary heap (i.e. the key is the excess value when a vertex is put on the heap: at the beginning of a scaling phase or when its excess becomes positive again), which is inspired by rules for max-flow push-relabel like largest excess selection [6] or excess scaling [7]. Every two scaling phases of COS, *vertex reordering* takes the order in which a BFS would visit the vertices and uses that to reorder arrays like potentials, excesses, or the CSR to improve the (average) cache locality when searching for a path to push flow. In COS, it can happen that an optimum flow is found but not recognised, which leads to more phases whose work consists of global potential updates and flow fluctuations. The division of costs by the GCD is also to reduce the number of those phases.

For SSP and CAS, we can use, apart from the standard binary heap, a radix heap and a hot queue [12]. We can also turn on multi-deficit augmentation where the Dijkstra search finds several augmenting paths at once and blocking augmentation which generalises Dinic's algorithm [19] to minimum cost flows. Every few scaling phases, *vertex renumbering* can rebuild the graph representation according to the BFS order, similar to the vertex renumbering in COS. When Dijkstra looks at a vertex's neighbours, their numerical parameters like potentials are more likely to be stored in the same cache line.

Verified Safeguard. We implement the verified safeguard in Isabelle/HOL in so-called Imperative HOL, verify it with the Refinement Framework [36] and export to SML to obtain verified, executable code. We reuse the same main graph format as for the oracles which is edges and vertices identified with indices in arrays. For every edge, we again have arrays of its first and second endpoint, respectively. We serialise the indices to 64-bit integers and the numerical values like costs to infinite-size/arbitrary-precision integers in SML.

The collection of oracles is a functional argument to the verified code, whose correctness proofs only assume the length of the arrays that contain the flow and its potentials or the infeasibility verdict and its cut, respectively. These are the only unproven, yet correctness-critical assumptions in the whole pipeline. We have an unverified SML-adaptor between the generated code and the oracles in C/C++ to convert the number formats. It specifically ensures with simple (and hence unlikely to-be-wrong) code that the arrays have the right length. The verified code first reduces to LBNF, calls the oracles, runs the checker and calls the cleanup, if necessary.

The verified checker receives the adaptor's answer and branches on its verdict: (1) If optimality is claimed, we iterate over the edges and check both feasibility of the flow f and CSl for the flow and the potentials π. For COS, we check approximate scaled CSl, e.g. $\alpha \cdot n \cdot c(e) + \pi(u) - \pi(v) \geq -1$ if $f(e) = 0 < u(e)$ for an edge going from u to v. Checking the balance constraints is trivial. (2) If infeasibility is claimed, we iterate over the indicator array encoding the supply side of the cut to compute the total balance. We accept the certificate if the supply in the cut is greater than the capacity, which is found by a pass over all edges.

If the certificate fails, the verified cleanup is called to transform the candidate flow into an optimum flow or infer infeasibility. A trivial step is truncating the flow values to within 0 and $u(e)$ for every edge e. We now have to use a flow algorithm with reasonably good running time and warm starting properties. We could use path augmentation to even out unsatisfied balance constraints. This method maintains the absence of cycles in the residual graph with negative costs as invariant which would first have to be established by cycle cancelling. According to Kovács [35], they show mostly inferior or even the worst performance, respectively. COS is practically efficient but relies on duals which cannot be initialised trivially. Also, the flow can oscillate a lot and thus, it does not make use of the candidate flow's proximity to the optimum (which is a reasonable assumption despite the discovered bug).

The NS is often of similar efficiency and every step does at least not take the flow further from the optimum. For an acyclic flow, i.e. a flow whose edges with flow strictly between 0 and their capacity form no undirected cycle [34], we can initialise NS's spanning tree solution with the big M method, including the initial potentials which are found by an exploration of the acyclic flow. One complication is that the flow must first be made acyclic, which can be done by a DFS-based procedure. Its cubic worst-case time is unlikely to be hit since a flow produced by a combinatorial algorithm would probably still be mostly acyclic despite the bug. Another reason why we use the NS for the cleanup instead of COS is that no step, not even a step in the initialisation, can move us further from the optimum, which is already close under realistic conditions.

We verify the graph's CSR representation and use that to verify the acyclification of the candidate flow. For the potentials, we symbolically compute with pairs where the first component is the number of big Ms in the potential and the second component is the 'ordinary' part of the potential. Although SML supports infinite-size integers, we should never actually evaluate a big M expression for performance reasons. As pivot rule, we use a verified implementation of block search with a candidate cache, which is also LEMON's default. Noteworthy is our verification of the classical tree structure that supports the deletion of an edge followed by reconnection of the two components into a new tree. For iterating over a component, it uses the *thread* (in Section 2.3.3 of TAOCP 1 [33] also called the *preorder sequential representation*), which is maintaining pointers to the next vertex in the pre-order iteration. Operations changing this tree representation are notoriously hard to implement, let alone to verify.

4 Evaluation

Setup. To evaluate the performance of our compound solver, we use the recommended LEMON as the baseline. The minimum cost flow problem as assumed by LEMON has different balance semantics, which are internally reduced to the equality semantics that we consider here by adding additional edges with a dummy vertex. If the balances sum up to 0, both semantics are identical, but LEMON would still perform its reductions and extend the graph. Since our LEMON-based oracles skip LEMON's reduction step, we compare our solver with both raw LEMON and a modified one that also skips the reduction to keep the comparison fair.

We compile the C++ oracles with `gcc -O3`, the SML code with `mlton` under default optimisation, and link the parts into one executable binary. We generally measure the time from after parsing the input file until before the printout of the solution ('compute'). For our solver, we additionally measure the time consumed by the oracle's actual solve, the certificate generation and the verified code (reduction to LBNF and check), with the remainder being glue between C++ and SML like number format conversion. We time out at one hour if the number of nodes is $< 1M$ and at two hours if it is $\geq 1M$.

Cost of Verification. We analyse the cost of verification on all core benchmark sets plus a few more (Appendices A.1 and A.2) on an AMD EPYC 9554 with all data fitting into the RAM.

We configure the solver's oracle to work like the baseline, i.e. with the same heuristics. For the *relative cost of verification (RCV)*, we take a baseline's best compute as the denominator and the solver's compute with the same algorithm as oracle as the numerator. It (very roughly) follows $\beta + (m \cdot 144\,ns + n \cdot 71\,ns)/T$ where β is the ratio between a baseline's and the oracle's compute time, and T is a baseline's compute time. β is around 1 because we have the same control flow; the only problem is that we cannot keep the points of measurement in code fully analogous. T depends on the best algorithm and the family. It grows roughly with $c \cdot m^\alpha$ where $\alpha > 1$[1] if we increase the number of vertices n at fixed density, and (usually) $\alpha < 1$ if we increase the density at a fixed n. Otherwise, α is family-dependent. Details about this model can be found in Appendix A.3.

The tables in Appendices A.1 and A.2 show that the RCV mostly decreases with the size of the graph if the density is fixed, consistent with the model. The ratio is usually ≤ 1.5 and never worse than 2.6, which happens very rarely and is still well within an order of magnitude. We can hence say that our architecture scales well overall. Sometimes the RCV even drops below 1, which happens almost exclusively for COS. For the global update and price refinement, LEMON uses buckets to group the vertices. Both scan twice over all buckets in LEMON, whereas our oracle performs one scan per heuristic only up to the maximum bucket index currently used. Peculiarly, the ratio sometimes starts increasing again a bit, most notably for `grid_square` and `grid_long` (Tables 7 and 8). We are not entirely sure, but the data suggests that the verified code becomes a bit more expensive and that the relative running time of the oracles also increases slightly. We have no explanation for this; it could be a memory locality effect.

With increasing density, we do not change the amount of flow to be moved; we rather increase the number of possibilities. This explains the sublinear growth for all algorithms except CAS, where we have superlinear growth because of many unsuccessful Dijkstra searches. Absolute verification costs grow linearly, however, leading to an increasing RCV.

As a miscellaneous observation, we note that there is no big difference between the baselines, even though the graph that pure LEMON works on has n additional edges. For COS, the verified code is more expensive, whereas there is little difference for the other algorithms: the certificate of scaled approximate optimality (= optimality for integers) can easily exceed 64 bits and is therefore returned in 128 bits, which leads to memory allocations for SML's arbitrary-precision integers. We cannot see any direct effect of the graph's shape on the RCV.

Heuristics. We evaluate the heuristics on the best two algorithms for every family on an AMD EPYC 7343 with all data fitting into the RAM. The comparison metric is the running time of the initialisation plus the main loop in the oracles. We also take those heuristics with a significant positive effect and evaluate their combination. Details are given in Appendix B.

We implemented a large number of heuristics and, as expected, the effect is most often inconclusively small or even adverse. The IMN pivot rule is a very clear loss. COS's BAR with approximate max-selection is a loss. No meaningful effect is observed for the order reversals and the presolving steps. Multi-deficit augmentation also remains silent, whereas the Dinic-like blocking flow for SSP and CAS is a heavy loss. Contrary to Kovács' conjecture [35], arc fixing is mostly harmful, with only a few weakly positive effects (Tables 29 and 31).

For a few heuristics, however, we have more decisive results. Using a radix heap for SSP's and CAS's Dijkstra can make the oracle 20%, in rare cases even 40%, faster (Tables 27 and 34 to 40), in contrast to the standard binary heap and the hot queue [12]. We think the radix heap's advantage is the fast append to a bucket, but it is strongest on single-source single-target instances. Probably, the number of nodes to check before reaching a target is larger there, which leads to more heap operations and increases the heap's leverage. One may compare instances with one source and target (Tables 27 and 34 to 36) with instances with several sources and

[1] This is not COS's α from the previous section.

targets (Tables 37 to 40).

Turning the arc mixing heuristic off for NS shows a benefit for dense graphs (Tables 23, 28 and 53) and is also beneficial for the vision instances (Tables 41, 43 and 55), but it harms sparse grid graphs with low degrees (Tables 30 and 32). As a partial explanation, we note that the latter have high order-locality in the input files, which is probably bad for pivot quality. Source and sink are far apart, requiring global pivots. For vision, we also have high locality, which exploits the cache, but sources and sinks are close, which could make the pivot less important.

For COS's path length in PAR, we also evaluate lengths 2 and 8, where LEMON's default length is 4. Shorter paths tend to be better for dense instances (Tables 24, 29, 48 and 54), whereas longer ones seem to benefit sparser ones (Tables 22, 26, 33, 46 and 52), although the evidence is not uniform (Tables 31, 42 and 56).

Concerning the stacking of heuristics, we observe that we have the best composition for SSP, CAS and NS, where the beneficial heuristics touch independent mechanisms (Tables 37 to 41 and 43). The beneficial heuristics for COS show a rough tendency towards interference and mutual neutralisation (Tables 22, 29, 52 and 54).

5 Discussion

This work is still ongoing, and what we present is a proof of concept: with careful engineering, a solver whose result is guaranteed to be correct can be as fast as an unverified one, although achieving parity on all instance families still requires work. The core of the approach is that the speed of the unverified solver and the guarantee given by the verified safeguard are orthogonal to each other. Accordingly, the architecture almost encourages experiments where new heuristics can be added and the oracles can be exchanged completely without affecting the correctness of the result. The approach as it stands works wherever certificates can be checked efficiently and a verified solver can be warm-started efficiently. Maximum flows and unweighted general matching satisfy both conditions, whereas for weighted general matching warm starting is harder.

AI usage. We have used generative AI to assist with developing the formal proofs, programming the unverified C++ oracles, as well as the design and the implementation of the experiments. The whole development was carried out with two Claude Code subscriptions at 100 dollars a month and one at 20 dollars a month.

Literature review. The general idea of combining an unverified solver with a verified certificate checker it is not new, and our approach is a variant of it. A similar approach was used for variants of the policy and value iteration algorithms for solving MDPs [48, 49] as well as for automated planning algorithms [51, 5]. An earlier similar approach combined a fast inexact solver for linear programs with an exact repairer [17]. To our knowledge, the combination of verified encoding, certificate checking and verified cleanup in one pipeline, and its application to minimum cost flows, is new.

Future work. Minimum cost flows are only the starting point, and we expect the approach to apply to many of the problems that are of interest to the DIMACS challenges, as long as a reasonable certificate exists, which usually requires some form of duality. Flows are a good first target in this respect, since their certificates are classical and easy to interpret and since they allow warm starts, whereas for matching, in particular general unweighted matching, the certificates are mathematically known but harder to interpret, and a warm start is not always available. Further extensions we intend to pursue are probabilistic oracles and oracles obtained by machine learning. On the implementation side, Goldberg's implementation of cost scaling [23] appears to be superior to the one we currently use, and we intend to add it as another oracle. Also, our verified code is currently Imperative HOL exported to Standard ML, and refining it further down to verified LLVM [37] would reduce the overhead that the verified parts still add.

References

[1] 13th DIMACS Implementation Challenge: Network Flows 2.0. Minimum-Cost Flow: Definition and Format. `https://coral.ise.lehigh.edu/flow-challenge-2-0/minimum-cost-flow-input-format/`, 2026. Accessed: 2026-09-17.

[2] Mohammad Abdulaziz. Graph Algorithms. In *Functional Data Structures and Algorithms: A Proof Assistant Approach*, volume 67, pages 273–298. Association for Computing Machinery, New York, NY, USA, 1 edition, September 2025.

[3] Mohammad Abdulaziz and Thomas Ammer. A Formal Analysis of Capacity Scaling Algorithms for Minimum Cost Flows. In *15th Int. Conf. Interact. Theorem Proving ITP*, 2024.

[4] Mohammad Abdulaziz, Thomas Ammer, Shriya Meenakshisundaram, and Adem Rimpapa. A Formal Analysis of Algorithms for Matroids and Greedoids. In *ITP*, 2025.

[5] Mohammad Abdulaziz and Friedrich Kurz. Formally verified SAT-Based AI planning. In *AAAI*, 2023.

[6] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, Englewood Cliffs, NJ, 1993.

[7] Ravindra K. Ahuja and James B. Orlin. A Fast and Simple Algorithm for the Maximum Flow Problem. *Operations Research*, 37(5):748–759, 1989.

[8] Eyad Alkassar, Sascha Böhme, Kurt Mehlhorn, and Christine Rizkallah. A Framework for the Verification of Certifying Computations. *J. Autom. Reason.*, 52(3):241–273, 2014.

[9] Heiko Becker, Nikita Zyuzin, Raphaël Monat, Eva Darulova, Magnus O. Myreen, and Anthony Fox. A Verified Certificate Checker for Finite-Precision Error Bounds in Coq and HOL4. In *Formal Methods in Computer Aided Design (FMCAD)*, pages 1–10, 2018.

[10] Yves Bertot. A Short Presentation of Coq. In *TPHOLs*, 2008.

[11] Robert G. Busacker and Paul J. Gowen. A Procedure for Determining a Family of Minimum-Cost Network Flow Patterns. Technical Report 15, Operations Research Office, Johns Hopkins University, 1960.

[12] Boris V. Cherkassky, Andrew V. Goldberg, and Craig Silverstein. Buckets, Heaps, Lists, and Monotone Priority Queues. *SIAM Journal on Computing*, 28(4):1326–1346, 1999.

[13] William H. Cunningham. Theoretical Properties of the Network Simplex Method. *Mathematics of Operations Research*, 4(2):196–208, 1979.

[14] George B. Dantzig. Application of the Simplex Method to a Transportation Problem. In Tjalling C. Koopmans, editor, *Activity Analysis of Production and Allocation*, pages 359–373. Wiley, New York, 1951.

[15] Leonardo de Moura and Sebastian Ullrich. The Lean 4 Theorem Prover and Programming Language. In *CADE*, 2021.

[16] Balázs Dezső, Alpár Jüttner, and Péter Kovács. LEMON – an open source C++ graph template library. *Electronic Notes in Theoretical Computer Science*, 264(5):23–45, 2011.

[17] Marcel Dhiflaoui, Stefan Funke, Carsten Kwappik, Kurt Mehlhorn, Michael Seel, Elmar Schömer, Ralph Schulte, and Dennis Weber. Certifying and repairing solutions to large LPs: How good are LP-solvers? In *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 2003)*, pages 255–256, Baltimore, Maryland, USA, 2003. ACM/SIAM.

[18] Edsger W Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1959.

[19] Yefim Dinitz. Algorithm for Solution of a Problem of Maximum Flow in Networks with Power Estimation. *Soviet Math. Dokl.*, 11:1277–1280, 01 1970.

[20] Jack Edmonds and Richard M. Karp. Theoretical Improvements in Algorithmic Efficiency for Network Flow Problems. *J. ACM*, 19(2):248–264, 1972.

[21] L. R. Ford and D. R. Fulkerson. *Flows in Networks*. Princeton University Press, 1962.

[22] Fred Glover, Darwin Klingman, and Joel Stutz. Augmented Threaded Index Method for Network Optimization. *INFOR: Information Systems and Operational Research*, 12(3):293–298, 1974.

[23] Andrew V. Goldberg. An Efficient Implementation of a Scaling Minimum-Cost Flow Algorithm. *Journal of Algorithms*, 22(1):1–29, 1997.

[24] Andrew V. Goldberg. The Partial Augment-Relabel Algorithm for the Maximum Flow Problem. In *16th Annual European Symposium on Algorithms (ESA)*, volume 5193 of *Lecture Notes in Computer Science*, pages 466–477. Springer, 2008.

[25] Andrew V. Goldberg and Robert E. Tarjan. A New Approach to the Maximum-Flow Problem. *Journal of the ACM*, 35(4):921–940, 1988.

[26] Andrew V. Goldberg and Robert E. Tarjan. Finding Minimum-Cost Circulations by Successive Approximation. *Mathematics of Operations Research*, 15(3):430–466, 1990.

[27] Donald Goldfarb, Jianxiu Hao, and Sheng-Roan Kai. Anti-Stalling Pivot Rules for the Network Simplex Algorithm. *Networks*, 20(1):79–91, 1990.

[28] C. A. R. Hoare. Proof of correctness of data representations. *Acta Informatica*, 1(4):271–281, December 1972.

[29] William S. Jewell. Optimal Flow Through Networks. Interim Technical Report 8, Massachusetts Institute of Technology, Cambridge, MA, 1958.

[30] Ellis L. Johnson. Networks and Basic Solutions. *Operations Research*, 14(4):619–623, 1966.

[31] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood. seL4: Formal verification of an OS kernel. In *SOSP*, 2009.

[32] Morton Klein. A Primal Method for Minimal Cost Flows with Applications to the Assignment and Transportation Problems. *Management Science*, 14(3):205–220, 1967.

[33] Donald E. Knuth. *The Art of Computer Programming, Volume 1: Fundamental Algorithms*. Addison-Wesley, Reading, MA, 3rd edition, 1997.

[34] Bernhard Korte and Jens Vygen. *Combinatorial Optimization*, volume 21 of *Algorithms and Combinatorics*. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[35] Péter Kovács. Minimum-Cost Flow Algorithms: An Experimental Evaluation. *Optimization Methods and Software*, 30(1):94–127, 2015.

[36] Peter Lammich. Refinement to Imperative HOL. *J. Autom. Reason.*, 62(4):481–503, 2019.

[37] Peter Lammich. Refinement of Parallel Algorithms down to LLVM. In *ITP*, 2022.

[38] Peter Lammich and S. Reza Sefidgar. Formalizing Network Flow Algorithms: A Refinement Approach in Isabelle/HOL. *J. Autom. Reason.*, 62(2):261–280, 2019.

[39] Xavier Leroy. Formal verification of a realistic compiler. *Commun. ACM*, 52(7):107–115, 2009.

[40] Ross M. McConnell, Kurt Mehlhorn, Stefan Näher, and Pascal Schweitzer. Certifying Algorithms. *Computer Science Review*, 5(2):119–161, 2011.

[41] Kurt Mehlhorn and Stefan Näher. From Algorithms to Working Programs: On the Use of Program Checking in LEDA. In *MFCS*, 1998.

[42] Kurt Mehlhorn and Stefan Näher. *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, United Kingdom, 1999.

[43] Robin Milner. Logic for Computable Functions Description of a Machine Implementation. Technical report, Stanford University, 1972.

[44] Tobias Nipkow, editor. *Functional Data Structures and Algorithms: A Proof Assistant Approach*, volume 67. Association for Computing Machinery, New York, NY, USA, 1 edition, August 2025.

[45] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL - A Proof Assistant for Higher-Order Logic*, volume 2283 of *Lecture Notes in Computer Science*. Springer, Germany, 2002.

[46] Alex Orden. The Transhipment Problem. *Management Science*, 2(3):276–285, 1956.

[47] James B. Orlin. A Faster Strongly Polynomial Minimum Cost Flow Algorithm. *Oper. Res.*, 41(2):338–350, April 1993.

[48] Maximilian Schäffeler and Mohammad Abdulaziz. Formally Verified Solution Methods for Infinite-Horizon Markov Decision Processes. In *AAAI*, 2023.

[49] Maximilian Schäffeler and Mohammad Abdulaziz. Formally Verified Approximate Policy Iteration. In *38th AAAI Conf. Artif. Intell. AAAI*, 2025.

[50] Jeremy G. Siek, Lie-Quan Lee, and Andrew Lumsdaine. *The Boost Graph Library: User Guide and Reference Manual*. Addison-Wesley, 2002.

[51] David Wang and Mohammad Abdulaziz. Formally Verified Certification of Unsolvability of Temporal Planning Problems. In *KRPlan Knowl. Represent. Meets Autom. Plan.*, 2025.

[52] Niklaus Wirth. Program Development by Stepwise Refinement. *Commun. ACM*, 14(4):221–227, 1971.

A The Cost of Verification

This appendix reports the raw measurements, one table per benchmark family and four rows per instance, one row for each of the algorithms NS, COS, CAS and SSP. The columns are the same in every table and are explained here once.

All numbers are *compute times*, i.e. we measure everything that happens after the input file has been parsed and before the output file is printed, so neither reading an instance nor writing a result is counted. The *Unit* column gives the unit of all times in the respective instance's rows, except for the two columns that state their own unit. A – marks a run that was not attempted and DNF one that did not finish.

We compare against two baselines. *Pure* is a mostly unmodified run of LEMON, while *Adapted* is a version of LEMON that we modified in a way one could expect from a reasonable programmer, so that our solver is not measured against a needlessly weak point of comparison. For each baseline, *Total* is its entire compute time and *Init+loop* the part of that time that is spent in the initialisation and in the main loop of the algorithm. For each instance, the best result that any baseline achieved, i.e. the smallest of the eight baseline *Total* times in that instance's four rows, is set in bold, and the ratio in the very row that holds this best baseline is boxed, as this is the ratio that is effectively achieved against the strongest available point of comparison. This is the RCV discussed in the evaluation section. The *Verified pipeline* columns break our solver's compute time down into the oracle run (*Oracle*), the emission of the certificate (*Cert.*) and the verified check (*Check*); *Total* is the solver's entire compute time and therefore exceeds the sum of those three. Finally, *Ratio* is, for one instance and one algorithm, the solver's compute time divided by the better of the two baselines' compute times for that same instance and algorithm, so a value below 1 means that our solver was faster than both baselines. Appendix A.3 distils these tables into the parameters of the cost model used in the evaluation.

A.1 Core Benchmarks

Table 1: Benchmark family `netgen_8`: A node sweep at the fixed density $m = 8n$. The running parameter k of the first column is the binary logarithm of the nominal number of nodes.

					Baselines				Verified pipeline				
					Pure		Adapted						
k	n	m	Unit	Alg.	Total	Init+loop	Total	Init+loop	Oracle	Cert. [ms]	Check [ms]	Total	Ratio
08	256	2048	ms	NS	**0.466**	0.415	0.547	0.411	0.431	0.010	0.165	0.805	1.729
				COS	1.272	1.124	1.353	1.120	1.134	0.021	0.187	1.614	1.270
				CAS	8.402	8.291	8.663	8.472	8.047	0.026	0.163	8.497	1.011
				SSP	3.770	3.655	3.953	3.760	3.541	0.020	0.163	3.980	1.056
11	2048	16 384	ms	NS	**6.534**	6.154	7.115	6.097	6.276	0.042	1.503	8.653	1.324
				COS	12.589	11.344	12.992	11.258	11.270	0.191	1.629	14.361	1.141
				CAS	495.990	494.990	489.370	487.810	474.090	0.184	1.532	477.140	0.975
				SSP	143.850	142.880	145.070	143.460	140.680	0.210	1.519	143.740	0.999
12	4096	32 768	ms	NS	**17.207**	16.431	18.630	16.501	16.820	0.196	2.102	21.612	1.256
				COS	29.804	27.220	30.669	26.991	26.811	0.443	2.358	33.027	1.108
				CAS	976.960	975.010	987.330	984.110	972.750	0.383	2.107	978.720	1.002
				SSP	540.690	538.670	545.620	542.350	538.080	0.376	2.104	544.010	1.006
13	8192	65 536	s	NS	**0.050**	0.048	0.053	0.048	0.048	0.398	4.199	0.056	1.120
				COS	0.063	0.057	0.064	0.057	0.056	0.896	4.675	0.067	1.065
				CAS	6.051	6.047	6.118	6.111	6.071	0.636	4.201	6.081	1.005
				SSP	2.022	2.018	2.034	2.028	2.018	0.727	4.201	2.028	1.003
14	16 384	131 072	s	NS	0.143	0.140	0.149	0.140	0.136	0.860	8.458	0.152	1.060
				COS	**0.136**	0.124	0.140	0.124	0.121	1.748	9.430	0.143	1.049
				CAS	25.752	25.743	26.042	26.027	25.915	1.114	8.373	25.935	1.007
				SSP	5.582	5.573	5.612	5.597	5.607	1.140	8.399	5.628	1.008
15	32 768	262 144	s	NS	0.425	0.419	0.422	0.403	0.406	1.649	17.105	0.438	1.038
				COS	**0.304**	0.278	0.310	0.274	0.270	2.719	19.084	0.314	1.032
				CAS	99.266	99.244	99.631	99.598	100.420	2.228	17.155	100.460	1.012
				SSP	20.294	20.272	20.395	20.362	20.441	2.338	17.145	20.483	1.009
16	65 536	524 288	s	NS	1.187	1.174	1.183	1.144	1.127	2.656	35.947	1.191	1.007
				COS	**0.698**	0.637	0.702	0.623	0.611	5.004	40.103	0.701	1.004
				CAS	548.550	548.500	546.590	546.520	553.010	3.872	36.426	553.100	1.012
				SSP	64.343	64.294	63.956	63.884	64.460	4.186	37.170	64.553	1.009
17	131 072	1 048 576	s	NS	3.591	3.564	3.557	3.472	3.402	5.007	76.391	3.535	0.994
				COS	**1.681**	1.539	1.689	1.505	1.459	9.625	85.368	1.652	0.983
				CAS	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
				SSP	208.740	208.620	207.990	207.830	209.100	7.909	76.443	209.300	1.006
18	262 144	2 097 152	s	NS	11.743	11.688	11.739	11.560	11.657	11.959	162.510	11.942	1.017
				COS	**4.651**	4.283	4.680	4.214	3.964	19.656	178.860	4.390	0.944
				CAS	–	–	–	–	–	–	–	–	–
				SSP	691.160	690.840	689.720	689.290	702.740	14.889	164.140	703.180	1.020
19	524 288	4 194 304	s	NS	36.286	36.174	36.555	36.157	36.015	19.798	350.890	36.596	1.009
				COS	**10.371**	9.492	10.625	9.483	8.656	39.371	391.430	9.584	0.924
				CAS	–	–	–	–	–	–	–	–	–
				SSP	2114.300	2113.500	2164.900	2163.800	2305.300	27.537	352.130	2306.300	1.091
20	1 048 576	8 388 608	s	NS	105.850	105.570	105.110	104.200	103.860	49.045	771.910	105.130	1.000
				COS	**25.339**	23.342	25.748	23.109	21.609	75.933	872.630	23.642	0.933
				CAS	–	–	–	–	–	–	–	–	–
				SSP	–	–	–	–	–	–	–	–	–
21	2 097 152	16 777 216	s	NS	360.710	360.080	361.830	359.340	346.820	69.905	2251.000	349.990	0.970
				COS	**56.487**	52.262	58.204	52.093	49.145	159.210	2540.000	54.136	0.958
				CAS	–	–	–	–	–	–	–	–	–
				SSP	–	–	–	–	–	–	–	–	–
22	4 194 304	33 554 432	s	NS	1734.300	1732.700	1755.000	1748.500	1780.600	137.060	6961.000	1789.500	1.032
				COS	**133.150**	123.940	135.600	121.930	116.670	302.280	8026.000	130.040	0.977
				CAS	–	–	–	–	–	–	–	–	–
				SSP	–	–	–	–	–	–	–	–	–

Table 2: Benchmark family `netgen_sr`: A node sweep at the density $m = n^{1.5}$. The running parameter k of the first column is the binary logarithm of the nominal number of nodes.

					Baselines				Verified pipeline				
					Pure		Adapted						
k	n	m	Unit	Alg.	Total	Init+loop	Total	Init+loop	Oracle	Cert. [ms]	Check [ms]	Total	Ratio
08	256	4096	ms	NS	**0.618**	0.528	0.789	0.530	0.561	0.011	0.282	1.152	1.862
				COS	1.715	1.442	1.867	1.447	1.479	0.024	0.328	2.290	1.335
				CAS	21.102	20.897	21.947	21.573	20.007	0.037	0.284	20.736	0.983
				SSP	5.352	5.148	5.700	5.334	4.953	0.022	0.286	5.666	1.059
09	512	11 585	ms	NS	**2.013**	1.754	2.549	1.753	1.829	0.014	1.044	3.467	1.722
				COS	5.464	4.671	5.891	4.690	4.823	0.074	1.159	6.940	1.270
				CAS	182.930	182.330	189.710	188.600	173.560	0.078	1.063	175.620	0.960
				SSP	28.682	28.085	30.047	28.962	27.075	0.045	1.036	29.067	1.013
10	1024	32 768	ms	NS	**5.597**	4.856	6.844	4.783	4.967	0.172	2.873	9.569	1.710
				COS	13.183	10.983	14.508	11.108	11.734	0.336	3.124	17.477	1.326
				CAS	575.130	573.440	590.850	587.830	549.710	0.415	2.797	555.250	0.965
				SSP	105.540	103.760	110.160	107.070	100.780	0.398	2.856	106.410	1.008
11	2048	92 682	s	NS	**0.016**	0.015	0.021	0.015	0.015	0.389	7.971	0.027	1.660
				COS	0.040	0.034	0.045	0.035	0.035	0.993	8.790	0.051	1.266
				CAS	6.731	6.725	6.843	6.833	6.434	0.873	7.993	6.450	0.958
				SSP	0.471	0.466	0.486	0.476	0.462	0.884	7.951	0.478	1.015
12	4096	262 144	s	NS	**0.059**	0.052	0.071	0.053	0.053	1.227	22.769	0.088	1.496
				COS	0.113	0.091	0.128	0.094	0.095	2.718	25.085	0.141	1.247
				CAS	24.533	24.514	24.730	24.700	23.736	2.302	22.946	23.781	0.969
				SSP	1.933	1.914	1.978	1.947	1.911	1.955	22.671	1.955	1.012
13	8192	741 455	s	NS	**0.211**	0.194	0.249	0.194	0.195	2.997	43.078	0.276	1.306
				COS	0.389	0.312	0.425	0.320	0.320	6.140	50.177	0.434	1.116
				CAS	217.140	217.080	218.400	218.310	210.320	5.856	43.120	210.430	0.969
				SSP	7.804	7.740	7.936	7.835	7.630	5.265	43.246	7.740	0.992
14	16 384	2 097 152	s	NS	**0.781**	0.724	0.887	0.725	0.729	7.032	124.870	0.975	1.248
				COS	1.707	1.406	1.817	1.421	1.238	16.167	142.470	1.583	0.927
				CAS	1724.400	1724.100	1743.400	1743.000	1693.000	13.689	121.990	1693.300	0.982
				SSP	33.424	33.157	34.089	33.723	33.193	12.505	121.980	33.520	1.003
15	32 768	5 931 642	s	NS	**2.850**	2.666	3.142	2.595	2.672	20.698	352.470	3.343	1.173
				COS	5.997	4.866	6.440	4.954	3.883	44.985	409.050	4.903	0.818
				CAS	–	–	–	–	–	–	–	–	–
				SSP	131.800	130.750	135.330	133.940	133.820	42.784	354.180	134.820	1.023
16	65 536	16 777 216	s	NS	**11.199**	10.475	12.102	10.295	10.979	64.567	1107.000	13.254	1.184
				COS	18.945	15.392	20.420	15.646	12.297	122.750	1269.000	15.433	0.815
				CAS	–	–	–	–	–	–	–	–	–
				SSP	555.260	551.960	584.260	579.400	590.010	104.570	1106.000	593.110	1.068

Table 3: Benchmark family `netgen_deg`: A density sweep at the fixed node count $n = 4096$. The running parameter i of the first column is the binary logarithm of the average degree m/n.

					Baselines				Verified pipeline				
					Pure		Adapted						
i	n	m	Unit	Alg.	Total	Init+loop	Total	Init+loop	Oracle	Cert. [ms]	Check [ms]	Total	Ratio
01	4096	8192	ms	NS	**8.137**	7.818	8.547	7.862	7.785	0.021	0.649	9.152	1.125
				COS	23.340	22.431	22.846	21.780	20.698	0.119	0.696	22.459	0.983
				CAS	104.300	103.610	102.860	101.930	103.020	0.081	0.674	104.740	1.018
				SSP	123.960	123.280	122.130	121.250	123.320	0.084	0.675	125.040	1.024
02	4096	16 384	ms	NS	**10.870**	10.387	11.594	10.395	10.872	0.048	1.610	13.530	1.245
				COS	21.793	20.262	21.682	19.763	19.279	0.244	1.717	22.662	1.045
				CAS	288.940	287.830	285.090	283.390	286.300	0.198	1.619	289.590	1.016
				SSP	214.480	213.350	211.770	210.070	211.720	0.195	1.630	215.000	1.015
03	4096	32 768	ms	NS	**17.184**	16.421	18.656	16.539	16.838	0.198	2.098	21.560	1.255
				COS	29.804	27.254	30.627	26.994	26.784	0.434	2.364	33.029	1.108
				CAS	976.340	974.300	988.340	985.070	971.910	0.377	2.094	977.840	1.002
				SSP	539.980	538.020	545.870	542.660	537.230	0.370	2.115	543.180	1.006
04	4096	65 536	s	NS	**0.025**	0.024	0.028	0.024	0.024	0.330	4.065	0.031	1.253
				COS	0.043	0.038	0.046	0.038	0.038	0.803	4.623	0.048	1.122
				CAS	3.907	3.903	3.975	3.969	3.843	0.802	4.058	3.852	0.986
				SSP	0.928	0.924	0.941	0.934	0.911	0.720	4.059	0.921	0.992
05	4096	131 072	s	NS	**0.039**	0.036	0.045	0.036	0.036	0.628	11.500	0.054	1.398
				COS	0.065	0.055	0.071	0.056	0.056	1.445	12.689	0.079	1.220
				CAS	9.052	9.044	9.205	9.191	8.854	1.261	11.513	8.877	0.981
				SSP	1.271	1.263	1.301	1.287	1.265	1.123	11.485	1.287	1.013
06	4096	262 144	s	NS	**0.059**	0.053	0.071	0.053	0.052	1.209	22.665	0.088	1.480
				COS	0.114	0.091	0.128	0.094	0.094	2.672	25.125	0.141	1.240
				CAS	24.499	24.482	24.736	24.706	23.755	2.395	22.802	23.799	0.971
				SSP	1.934	1.916	1.978	1.947	1.910	1.939	22.762	1.954	1.010
07	4096	524 288	s	NS	**0.097**	0.084	0.121	0.084	0.084	2.292	45.308	0.158	1.628
				COS	0.233	0.183	0.258	0.190	0.191	4.465	49.606	0.283	1.217
				CAS	78.743	78.702	79.105	79.041	75.297	4.207	45.540	75.388	0.957
				SSP	3.422	3.381	3.514	3.449	3.335	3.434	45.151	3.424	1.001
08	4096	1 048 576	s	NS	**0.151**	0.123	0.201	0.123	0.123	4.204	89.847	0.274	1.811
				COS	0.520	0.408	0.574	0.423	0.409	8.522	99.437	0.601	1.156
				CAS	295.630	295.540	300.180	300.030	281.740	8.031	91.060	281.930	0.954
				SSP	4.717	4.619	4.925	4.784	4.564	6.091	89.780	4.744	1.006
09	4096	2 097 152	s	NS	**0.258**	0.191	0.360	0.192	0.190	8.442	180.710	0.495	1.919
				COS	1.489	1.194	1.612	1.227	1.058	15.968	197.170	1.453	0.976
				CAS	969.390	969.130	985.690	985.320	924.820	15.709	182.080	925.210	0.954
				SSP	8.249	7.987	8.725	8.373	7.854	12.662	178.830	8.231	0.998
10	4096	4 194 304	s	NS	**0.551**	0.361	0.774	0.362	0.339	15.822	357.780	0.956	1.734
				COS	3.611	2.889	3.909	2.977	2.309	32.088	396.450	3.146	0.871
				CAS	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
				SSP	14.794	14.140	16.186	15.295	13.945	26.181	356.860	14.718	0.995
11	4096	8 388 608	s	NS	**0.979**	0.591	1.456	0.587	0.575	29.638	716.390	1.833	1.872
				COS	7.956	6.383	8.566	6.510	5.100	56.371	797.470	6.782	0.853
				CAS	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
				SSP	27.491	26.037	29.633	27.671	26.008	48.768	708.570	27.572	1.003
12	4096	16 252 992	s	NS	**1.447**	0.934	2.371	0.944	0.916	53.510	1489.000	3.378	2.335
				COS	16.183	13.040	17.362	13.320	10.320	109.510	1638.000	13.569	0.838
				CAS	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
				SSP	54.739	51.788	58.652	54.789	51.289	115.120	1522.000	54.511	0.996

Table 4: Benchmark family `goto_8`: A node sweep at the fixed density $m = 8n$. The running parameter k of the first column is the binary logarithm of the nominal number of nodes.

					Baselines				Verified pipeline				
					Pure		Adapted						
k	n	m	Unit	Alg.	Total	Init+loop	Total	Init+loop	Oracle	Cert. [ms]	Check [ms]	Total	Ratio
08	256	2048	ms	NS	**0.663**	0.611	0.748	0.609	0.638	0.011	0.175	1.035	1.560
				COS	2.974	2.834	3.033	2.817	2.806	0.024	0.186	3.276	1.101
				CAS	5.608	5.507	5.670	5.491	5.391	0.025	0.177	5.857	1.044
				SSP	34.042	33.935	34.219	34.032	32.898	0.045	0.170	33.389	0.981
11	2048	16 384	s	NS	**0.023**	0.023	0.023	0.022	0.023	0.060	1.567	0.026	1.108
				COS	0.043	0.041	0.042	0.041	0.040	0.260	1.617	0.043	1.031
				CAS	0.135	0.134	0.134	0.133	0.132	0.205	1.572	0.135	1.005
				SSP	3.814	3.813	3.725	3.724	3.672	0.220	1.540	3.675	0.987
12	4096	32 768	s	NS	**0.113**	0.112	0.114	0.112	0.108	0.293	2.259	0.113	0.998
				COS	0.143	0.140	0.141	0.138	0.139	0.598	2.336	0.145	1.025
				CAS	2.095	2.093	2.060	2.057	2.054	0.362	2.246	2.060	1.000
				SSP	27.564	27.562	27.314	27.311	26.607	0.380	2.247	26.613	0.974
13	8192	65 536	s	NS	0.327	0.325	0.330	0.326	0.318	0.505	4.287	0.326	0.999
				COS	0.258	0.254	**0.257**	0.250	0.247	1.007	4.523	0.258	1.004
				CAS	1.289	1.285	1.266	1.261	1.253	0.619	4.263	1.263	0.998
				SSP	72.989	72.985	71.889	71.883	69.189	0.622	4.267	69.199	0.963
14	16 384	131 072	s	NS	1.221	1.218	1.234	1.225	1.271	0.823	8.435	1.286	1.054
				COS	0.696	0.687	**0.684**	0.670	0.655	1.736	9.005	0.676	0.988
				CAS	3.682	3.675	3.619	3.607	3.523	1.116	8.534	3.543	0.979
				SSP	293.890	293.880	289.260	289.250	274.510	1.319	8.477	274.530	0.949
15	32 768	262 144	s	NS	34.236	34.230	32.622	32.603	31.593	1.687	17.198	31.624	0.969
				COS	2.418	2.396	**2.380**	2.349	2.342	3.039	18.725	2.386	1.003
				CAS	189.050	189.030	187.340	187.320	185.270	2.286	17.217	185.310	0.989
				SSP	1282.700	1282.700	1277.800	1277.700	1270.300	2.304	17.230	1270.400	0.994
16	65 536	524 288	s	NS	38.356	38.344	37.191	37.155	35.666	3.083	34.091	35.729	0.961
				COS	5.594	5.552	**5.390**	5.328	5.140	5.875	36.263	5.225	0.969
				CAS	35.705	35.672	35.232	35.179	34.066	4.143	34.074	34.149	0.969
				SSP	3298.900	3298.900	3275.800	3275.800	3148.600	4.129	34.066	3148.700	0.961
17	131 072	1 048 576	s	NS	149.440	149.410	145.890	145.820	152.080	6.727	68.737	152.210	1.043
				COS	21.854	21.762	**20.960**	20.828	19.734	12.920	73.118	19.912	0.950
				CAS	115.670	115.600	114.260	114.150	105.990	8.020	68.861	106.160	0.929
				SSP	–	–	–	–	–	–	–	–	–
18	262 144	2 097 152	s	NS	3362.100	3362.100	3326.200	3326.000	3203.400	12.901	140.380	3203.700	0.963
				COS	42.454	42.238	**41.187**	40.890	41.971	25.762	154.930	42.363	1.029
				CAS	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
				SSP	–	–	–	–	–	–	–	–	–
19	524 288	4 194 304	s	NS	2386.400	2386.300	2393.000	2392.700	2413.900	24.851	289.840	2414.500	1.012
				COS	287.410	287.000	**282.310**	281.730	270.890	43.717	308.370	271.630	0.962
				CAS	1276.800	1276.500	1252.600	1252.000	1162.900	32.977	289.580	1163.600	0.929
				SSP	–	–	–	–	–	–	–	–	–
20	1 048 576	8 388 608	s	NS	–	–	–	–	–	–	–	–	–
				COS	DNF	DNF	DNF	DNF	1009.100	85.729	639.780	1010.700	DNF
				CAS	–	–	–	–	–	–	–	–	–
				SSP	–	–	–	–	–	–	–	–	–
21	2 097 152	16 777 216	s	NS	–	–	–	–	–	–	–	–	–
				COS	694.020	691.690	**679.570**	676.200	656.740	149.510	1489.000	660.310	0.972
				CAS	–	–	–	–	–	–	–	–	–
				SSP	–	–	–	–	–	–	–	–	–
22	4 194 304	33 554 432	s	NS	–	–	–	–	–	–	–	–	–
				COS	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
				CAS	–	–	–	–	–	–	–	–	–
				SSP	–	–	–	–	–	–	–	–	–

Table 5: Benchmark family `goto_sr`: A node sweep at the density $m = n^{1.5}$. The running parameter k of the first column is the binary logarithm of the nominal number of nodes.

					Baselines				Verified pipeline				
					Pure		Adapted						
k	n	m	Unit	Alg.	Total	Init+loop	Total	Init+loop	Oracle	Cert. [ms]	Check [ms]	Total	Ratio
08	256	4096	ms	NS	**1.586**	1.496	1.758	1.504	1.588	0.012	0.314	2.221	1.400
				COS	5.683	5.429	5.836	5.428	5.512	0.027	0.322	6.259	1.101
				CAS	22.784	22.595	22.996	22.642	21.806	0.037	0.303	22.552	0.990
				SSP	128.430	128.240	131.390	131.030	122.930	0.073	0.308	123.720	0.963
09	512	11 585	s	NS	**0.006**	0.006	0.007	0.006	0.006	0.017	1.111	0.008	1.284
				COS	0.019	0.019	0.020	0.019	0.019	0.091	1.142	0.021	1.100
				CAS	0.128	0.127	0.130	0.129	0.123	0.080	1.100	0.125	0.978
				SSP	1.189	1.188	1.210	1.209	1.127	0.079	1.103	1.130	0.950
10	1024	32 768	s	NS	**0.029**	0.029	0.031	0.029	0.029	0.199	3.020	0.034	1.157
				COS	0.065	0.063	0.066	0.063	0.066	0.498	3.098	0.071	1.100
				CAS	0.899	0.898	0.909	0.906	0.866	0.434	2.958	0.871	0.969
				SSP	12.650	12.649	12.766	12.763	11.901	0.395	3.056	11.907	0.941
11	2048	92 682	s	NS	**0.145**	0.143	0.149	0.143	0.144	0.600	8.367	0.157	1.086
				COS	0.223	0.218	0.229	0.219	0.229	1.189	8.652	0.246	1.099
				CAS	6.490	6.485	6.560	6.551	6.308	0.877	8.427	6.324	0.974
				SSP	123.940	123.940	125.190	125.180	116.070	0.938	8.439	116.090	0.937
12	4096	262 144	s	NS	0.843	0.836	0.860	0.841	0.766	1.341	23.956	0.803	0.953
				COS	**0.700**	0.682	0.711	0.683	0.723	2.720	24.508	0.769	1.098
				CAS	36.954	36.940	36.889	36.862	35.966	2.306	23.907	36.012	0.976
				SSP	1464.200	1464.200	1456.800	1456.700	1339.000	2.363	24.035	1339.100	0.919
13	8192	741 455	s	NS	3.898	3.881	3.931	3.877	3.702	3.200	46.350	3.787	0.972
				COS	**2.495**	2.440	2.551	2.462	2.638	6.913	48.294	2.752	1.103
				CAS	302.470	302.420	305.280	305.200	296.040	5.878	46.157	296.150	0.979
				SSP	–	–	–	–	–	–	–	–	–
14	16 384	2 097 152	s	NS	20.519	20.462	20.854	20.692	20.279	10.513	129.990	20.538	1.001
				COS	**10.877**	10.712	11.413	11.153	11.974	21.172	135.510	12.305	1.131
				CAS	2250.400	2250.300	2263.200	2263.000	2250.000	15.606	130.560	2250.300	1.000
				SSP	–	–	–	–	–	–	–	–	–
15	32 768	5 931 642	s	NS	266.900	266.710	263.760	263.210	250.040	28.041	369.290	250.750	0.951
				COS	**46.952**	46.489	47.979	47.171	50.081	47.839	383.920	51.001	1.086
				CAS	–	–	–	–	–	–	–	–	–
				SSP	–	–	–	–	–	–	–	–	–
16	65 536	16 777 216	s	NS	1141.700	1140.900	1114.500	1112.800	979.090	76.916	1094.000	981.380	0.881
				COS	**153.430**	151.960	156.940	154.300	170.420	121.610	1138.000	173.200	1.129
				CAS	–	–	–	–	–	–	–	–	–
				SSP	–	–	–	–	–	–	–	–	–

Table 6: Benchmark family `grid_wide`: Grids of width 16 and height $2^k/16$. The running parameter k of the first column is the binary logarithm of the nominal number of nodes.

					Baselines				Verified pipeline				
					Pure		Adapted						
k	n	m	Unit	Alg.	Total	Init+loop	Total	Init+loop	Oracle	Cert. [ms]	Check [ms]	Total	Ratio
08	258	512	ms	NS	**0.181**	0.154	0.216	0.154	0.163	0.008	0.072	0.405	2.239
				COS	0.873	0.818	0.876	0.799	0.765	0.033	0.089	1.051	1.204
				CAS	0.502	0.463	0.518	0.452	0.462	0.015	0.071	0.697	1.390
				SSP	0.832	0.793	0.854	0.785	0.799	0.016	0.073	1.039	1.249
11	2050	4208	ms	NS	**1.826**	1.676	2.028	1.669	1.728	0.014	0.362	2.512	1.376
				COS	11.332	10.961	11.313	10.837	10.267	0.077	0.387	11.221	0.992
				CAS	19.461	19.198	19.262	18.861	18.834	0.039	0.362	19.755	1.026
				SSP	70.444	70.180	69.904	69.490	70.467	0.071	0.364	71.422	1.022
12	4098	8432	ms	NS	**4.265**	3.985	4.753	3.998	4.101	0.020	0.691	5.536	1.298
				COS	30.464	29.738	30.375	29.387	27.461	0.153	0.732	29.249	0.963
				CAS	78.396	77.901	77.676	76.850	76.752	0.076	0.695	78.436	1.010
				SSP	321.190	320.710	316.720	315.900	319.300	0.083	0.691	321.000	1.014
13	8194	16 880	s	NS	**0.009**	0.008	0.010	0.008	0.009	0.128	1.323	0.011	1.273
				COS	0.076	0.074	0.075	0.073	0.067	0.380	1.426	0.071	0.945
				CAS	0.318	0.317	0.315	0.313	0.313	0.146	1.340	0.316	1.004
				SSP	1.422	1.421	1.402	1.400	1.423	0.150	1.346	1.427	1.018
14	16 386	33 776	s	NS	**0.020**	0.019	0.022	0.019	0.020	0.261	2.628	0.025	1.244
				COS	0.223	0.220	0.217	0.213	0.202	0.814	2.793	0.209	0.964
				CAS	1.398	1.396	1.380	1.376	1.409	0.352	2.663	1.416	1.026
				SSP	6.159	6.157	5.995	5.992	6.207	0.372	2.658	6.214	1.037
15	32 770	67 568	s	NS	**0.046**	0.044	0.050	0.044	0.046	0.761	5.194	0.057	1.222
				COS	0.504	0.498	0.496	0.488	0.452	1.472	5.547	0.465	0.938
				CAS	5.714	5.710	5.630	5.623	5.780	0.556	5.236	5.793	1.029
				SSP	25.559	25.555	25.171	25.165	25.945	0.475	5.238	25.958	1.031
16	65 538	135 152	s	NS	**0.108**	0.104	0.117	0.104	0.107	1.440	10.490	0.128	1.185
				COS	1.509	1.498	1.484	1.467	1.376	2.345	11.058	1.404	0.947
				CAS	24.304	24.296	23.969	23.955	25.395	1.039	10.558	25.424	1.061
				SSP	104.520	104.510	102.130	102.120	109.380	1.113	10.536	109.410	1.071
17	131 074	270 320	s	NS	**0.265**	0.256	0.283	0.257	0.262	2.364	21.265	0.305	1.147
				COS	4.543	4.521	4.350	4.316	3.827	4.808	22.431	3.883	0.893
				CAS	101.690	101.670	99.677	99.648	105.720	1.884	21.302	105.780	1.061
				SSP	420.150	420.140	411.050	411.020	438.910	1.875	21.275	438.960	1.068
18	262 146	540 656	s	NS	**0.677**	0.659	0.711	0.657	0.672	4.248	43.293	0.760	1.123
				COS	15.194	15.145	14.109	14.033	11.714	9.222	45.212	11.829	0.838
				CAS	443.720	443.690	435.690	435.630	455.650	2.753	43.471	455.770	1.046
				SSP	1701.600	1701.600	1662.600	1662.500	1727.900	3.518	42.413	1728.000	1.039
19	524 290	1 081 328	s	NS	**1.932**	1.895	2.007	1.897	1.937	8.988	85.337	2.109	1.092
				COS	57.704	57.605	54.210	54.067	46.089	21.292	90.407	46.320	0.854
				CAS	2078.700	2078.600	1958.100	1958.000	2129.600	6.617	85.154	2129.800	1.088
				SSP	–	–	–	–	–	–	–	–	–
20	1 048 578	2 162 672	s	NS	**5.538**	5.465	5.714	5.499	5.579	15.446	168.820	5.915	1.068
				COS	181.450	181.250	174.420	174.130	138.040	35.611	177.980	138.470	0.794
				CAS	–	–	–	–	–	–	–	–	–
				SSP	–	–	–	–	–	–	–	–	–

Table 7: Benchmark family `grid_square`: Square grids of side $2^{k/2}$. The running parameter k of the first column is the binary logarithm of the nominal number of nodes.

					Baselines				Verified pipeline				
					Pure		Adapted						
k	n	m	Unit	Alg.	Total	Init+loop	Total	Init+loop	Oracle	Cert. [ms]	Check [ms]	Total	Ratio
08	258	512	ms	NS	**0.186**	0.159	0.215	0.154	0.163	0.008	0.074	0.414	2.231
				COS	0.874	0.819	0.883	0.804	0.775	0.026	0.083	1.042	1.192
				CAS	0.511	0.473	0.520	0.454	0.461	0.015	0.079	0.705	1.379
				SSP	0.828	0.790	0.847	0.778	0.802	0.016	0.072	1.042	1.258
11	2027	4050	ms	NS	**3.094**	2.943	3.296	2.964	3.059	0.016	0.335	3.819	1.234
				COS	11.687	11.340	11.654	11.221	10.750	0.077	0.371	11.667	1.001
				CAS	11.182	10.938	11.157	10.813	10.811	0.033	0.339	11.660	1.045
				SSP	51.133	50.882	50.723	50.361	50.804	0.061	0.343	51.691	1.019
12	4098	8192	ms	NS	**8.992**	8.704	9.459	8.779	8.882	0.022	0.640	10.248	1.140
				COS	28.223	27.517	27.857	27.002	25.820	0.126	0.713	27.565	0.990
				CAS	29.972	29.476	30.041	29.288	29.044	0.051	0.648	30.629	1.022
				SSP	215.210	214.700	213.240	212.490	214.990	0.082	0.655	216.630	1.016
13	8283	16 562	ms	NS	**26.768**	26.204	28.132	26.646	26.028	0.154	1.250	28.688	1.072
				COS	75.102	73.771	74.152	72.251	68.655	0.363	1.384	72.030	0.971
				CAS	109.080	108.100	108.050	106.460	105.630	0.133	1.278	108.740	1.006
				SSP	933.340	932.390	921.240	919.650	927.840	0.151	1.271	931.000	1.011
14	16 386	32 768	s	NS	**0.095**	0.094	0.098	0.095	0.095	0.358	2.448	0.101	1.061
				COS	0.204	0.201	0.199	0.196	0.189	0.728	2.691	0.196	0.983
				CAS	0.319	0.317	0.315	0.312	0.308	0.177	2.473	0.314	0.999
				SSP	4.067	4.065	3.963	3.960	3.981	0.181	2.464	3.987	1.006
15	32 763	65 522	s	NS	0.586	0.583	0.596	0.590	0.380	0.865	4.843	0.390	0.666
				COS	0.529	0.523	**0.508**	0.501	0.475	1.431	5.317	0.488	0.961
				CAS	1.000	0.996	0.978	0.972	0.944	0.243	4.900	0.956	0.977
				SSP	16.932	16.928	16.487	16.481	16.420	0.225	4.907	16.432	0.997
16	65 538	131 072	s	NS	1.357	1.353	1.385	1.374	1.355	1.420	9.626	1.376	1.014
				COS	1.316	1.305	**1.280**	1.266	1.233	2.548	10.596	1.258	0.982
				CAS	3.014	3.006	2.968	2.956	2.913	0.364	9.692	2.936	0.989
				SSP	61.612	61.603	60.032	60.019	60.621	0.411	9.701	60.644	1.010
17	131 046	262 088	s	NS	6.270	6.260	6.296	6.273	6.470	2.596	19.309	6.511	1.038
				COS	3.972	3.949	**3.784**	3.754	3.595	4.675	21.073	3.644	0.963
				CAS	9.617	9.600	9.352	9.328	9.062	0.594	19.464	9.108	0.974
				SSP	243.860	243.840	236.510	236.490	237.640	0.654	19.444	237.690	1.005
18	262 146	524 288	s	NS	29.750	29.731	29.918	29.874	29.219	4.797	38.404	29.299	0.985
				COS	11.541	11.497	**10.696**	10.638	10.045	8.698	42.275	10.143	0.948
				CAS	33.082	33.050	32.304	32.256	31.160	1.040	38.818	31.252	0.967
				SSP	886.370	886.340	852.670	852.630	864.880	1.105	39.116	864.970	1.014
19	524 178	1 048 352	s	NS	132.810	132.770	133.770	133.670	130.750	10.738	78.017	130.910	0.986
				COS	31.776	31.682	**29.863**	29.747	29.152	20.840	84.724	29.355	0.983
				CAS	105.690	105.630	103.750	103.650	98.461	2.286	78.449	98.650	0.951
				SSP	3147.200	3147.200	3054.200	3054.100	3069.300	2.169	78.337	3069.500	1.005
20	1 048 578	2 097 152	s	NS	–	–	–	–	–	–	–	–	–
				COS	86.570	86.382	**82.762**	82.515	83.165	34.678	169.540	83.561	1.010
				CAS	400.840	400.710	393.100	392.900	375.530	3.581	155.650	375.890	0.956
				SSP	–	–	–	–	–	–	–	–	–

Table 8: Benchmark family `grid_long`: Grids of height 16 and width $2^k/16$. The running parameter k of the first column is the binary logarithm of the nominal number of nodes.

					Baselines				Verified pipeline				
					Pure		Adapted						
k	n	m	Unit	Alg.	Total	Init+loop	Total	Init+loop	Oracle	Cert. [ms]	Check [ms]	Total	Ratio
08	258	512	ms	NS	**0.179**	0.152	0.216	0.155	0.163	0.008	0.071	0.382	2.135
				COS	0.865	0.809	0.873	0.793	0.765	0.026	0.084	1.037	1.198
				CAS	0.500	0.460	0.517	0.452	0.468	0.015	0.072	0.706	1.413
				SSP	0.830	0.790	0.859	0.784	0.792	0.016	0.073	1.039	1.252
11	2050	3984	ms	NS	**2.836**	2.685	3.013	2.675	2.760	0.016	0.324	3.499	1.234
				COS	7.210	6.869	7.267	6.834	6.475	0.066	0.368	7.379	1.023
				CAS	3.765	3.524	3.849	3.484	3.513	0.025	0.320	4.372	1.161
				SSP	9.802	9.555	9.806	9.442	9.545	0.029	0.319	10.374	1.058
12	4098	7952	ms	NS	8.440	8.149	8.795	8.144	8.353	0.023	0.600	9.681	1.147
				COS	16.561	15.899	16.615	15.792	14.920	0.110	0.671	16.555	1.000
				CAS	**8.246**	7.751	8.375	7.679	7.691	0.033	0.600	9.190	1.115
				SSP	18.735	18.250	18.746	18.043	18.208	0.041	0.608	19.734	1.053
13	8194	15 888	ms	NS	26.304	25.727	27.068	25.770	26.127	0.038	1.147	28.573	1.086
				COS	39.135	37.797	38.859	37.285	35.341	0.222	1.297	38.390	0.988
				CAS	15.683	14.706	**15.661**	14.285	14.457	0.045	1.164	17.223	1.100
				SSP	32.877	31.902	32.534	31.167	31.518	0.060	1.147	34.347	1.056
14	16 386	31 760	ms	NS	95.978	94.821	97.831	95.064	93.983	0.305	2.246	98.905	1.030
				COS	94.238	91.619	93.412	90.039	85.801	0.645	2.537	91.922	0.984
				CAS	33.226	31.265	**33.127**	30.228	30.995	0.115	2.250	36.575	1.104
				SSP	58.454	56.471	57.813	54.822	55.975	0.148	2.259	61.626	1.066
15	32 770	63 504	ms	NS	362.570	360.370	365.650	360.310	362.980	0.812	4.440	372.860	1.028
				COS	179.940	174.320	178.740	171.570	164.520	1.420	5.013	176.800	0.989
				CAS	**51.647**	47.758	52.241	46.381	47.435	0.196	4.452	58.509	1.133
				SSP	86.930	83.117	86.863	81.017	82.983	0.235	4.474	94.127	1.084
16	65 538	126 992	s	NS	1.393	1.389	1.402	1.391	1.395	1.391	8.840	1.415	1.016
				COS	0.360	0.348	0.356	0.341	0.320	2.320	9.999	0.344	0.967
				CAS	**0.091**	0.083	0.092	0.081	0.083	0.337	8.868	0.105	1.153
				SSP	0.117	0.109	0.118	0.106	0.109	0.354	8.865	0.131	1.123
17	131 074	253 968	s	NS	5.351	5.342	5.374	5.352	5.372	2.412	17.620	5.411	1.011
				COS	0.814	0.793	0.792	0.764	0.712	4.156	19.984	0.759	0.959
				CAS	0.162	0.147	0.166	0.143	0.147	0.571	17.691	0.191	1.176
				SSP	**0.158**	0.142	0.162	0.139	0.143	0.570	17.648	0.187	1.186
18	262 146	507 920	s	NS	21.452	21.433	21.481	21.435	21.464	5.494	35.527	21.543	1.004
				COS	1.741	1.698	1.646	1.591	1.452	7.765	40.768	1.547	0.940
				CAS	0.311	0.280	0.318	0.272	0.281	1.001	35.391	0.367	1.180
				SSP	**0.305**	0.273	0.313	0.267	0.274	1.031	35.437	0.361	1.184
19	524 290	1 015 824	s	NS	86.017	85.978	86.145	86.053	86.308	10.477	70.759	86.465	1.005
				COS	4.063	3.978	3.898	3.786	3.470	15.515	80.152	3.658	0.938
				CAS	0.671	0.608	0.689	0.596	0.619	1.900	70.587	0.793	1.181
				SSP	**0.658**	0.596	0.679	0.587	0.607	1.840	70.514	0.781	1.186
20	1 048 578	2 031 632	s	NS	–	–	–	–	–	–	–	–	–
				COS	–	–	–	–	–	–	–	–	–
				CAS	1.590	1.465	1.673	1.488	1.557	3.497	140.910	1.904	1.198
				SSP	**1.569**	1.443	1.662	1.473	1.537	3.588	141.100	1.891	1.205

Table 9: Benchmark family `road_paths`: US state road network (real data). The first column lists the instances, ordered by increasing number of nodes n.

					Baselines				Verified pipeline				
					Pure		Adapted						
Instance	n	m	Unit	Alg.	Total	Init+loop	Total	Init+loop	Oracle	Cert. [ms]	Check [ms]	Total	Ratio
01_DC	9559	29 768	ms	NS	15.670	14.808	17.028	14.824	15.173	0.165	2.089	19.280	1.230
				COS	35.377	33.113	36.017	32.837	31.023	0.415	2.237	36.213	1.024
				CAS	6.455	4.752	7.321	4.646	4.752	0.164	2.078	9.650	1.495
				SSP	**6.372**	4.704	7.267	4.568	4.727	0.174	2.076	9.671	1.518
02_DE	49 109	120 576	ms	NS	144.770	140.810	150.260	140.390	143.380	1.232	9.109	161.770	1.117
				COS	268.130	257.740	266.280	252.350	239.270	2.253	9.266	261.460	0.982
				CAS	31.071	23.394	34.294	22.576	23.220	0.634	8.632	44.283	1.425
				SSP	**30.659**	22.985	33.870	22.250	23.242	0.617	8.833	44.330	1.446
03_NH	116 920	265 402	ms	NS	674.020	665.680	678.940	654.600	678.190	2.281	19.234	717.580	1.065
				COS	963.330	940.250	932.480	898.820	845.340	4.130	20.587	894.170	0.959
				CAS	93.965	77.050	102.660	74.041	76.736	1.474	19.257	124.740	1.328
				SSP	**92.651**	75.703	102.020	73.286	76.903	1.452	19.180	124.460	1.343
04_NV	261 155	620 924	s	NS	2.800	2.779	2.876	2.819	2.854	4.526	45.578	2.945	1.052
				COS	3.893	3.839	3.632	3.554	3.339	8.910	49.044	3.455	0.951
				CAS	0.321	0.281	0.336	0.270	0.280	2.984	45.952	0.393	1.225
				SSP	**0.320**	0.279	0.334	0.268	0.279	2.919	45.305	0.391	1.222
05_WI	519 157	1 266 876	s	NS	9.122	9.080	9.275	9.154	9.150	10.678	96.204	9.344	1.024
				COS	10.042	9.924	9.395	9.225	8.859	19.805	103.030	9.104	0.969
				CAS	1.244	1.158	1.227	1.085	1.121	6.271	96.718	1.368	1.115
				SSP	1.232	1.148	**1.223**	1.084	1.116	5.744	96.551	1.360	1.112
06_FL	1 048 506	2 653 624	s	NS	–	–	–	–	–	–	–	–	–
				COS	–	–	–	–	–	–	–	–	–
				CAS	**2.083**	1.901	2.098	1.787	1.842	12.543	211.290	2.375	1.140
				SSP	2.090	1.910	2.099	1.794	1.859	11.438	211.910	2.384	1.141
07_TX	2 073 870	5 156 088	s	NS	–	–	–	–	–	–	–	–	–
				COS	–	–	–	–	–	–	–	–	–
				CAS	7.295	6.947	7.141	6.515	6.688	21.355	427.240	7.727	1.082
				SSP	7.261	6.902	**7.110**	6.473	6.665	24.191	424.890	7.723	1.086

Table 10: Benchmark family `road_flow`: US state road network (real data). The first column lists the instances, ordered by increasing number of nodes n.

					Baselines				Verified pipeline				
					Pure		Adapted						
Instance	n	m	Unit	Alg.	Total	Init+loop	Total	Init+loop	Oracle	Cert. [ms]	Check [ms]	Total	Ratio
01_DC	9559	29 768	ms	NS	16.666	15.822	18.031	15.872	15.529	0.161	2.097	19.578	1.175
				COS	49.865	47.414	49.966	46.762	44.348	0.440	2.250	49.549	0.994
				CAS	**12.170**	10.416	12.800	10.185	10.473	0.176	2.085	15.404	1.266
				SSP	15.444	13.768	16.050	13.387	13.725	0.180	2.090	18.637	1.207
02_DE	49 109	120 576	ms	NS	144.250	140.290	150.070	140.120	142.820	1.229	9.205	161.320	1.118
				COS	374.200	363.910	369.330	355.750	335.560	2.209	9.468	358.040	0.969
				CAS	**72.582**	64.801	74.461	62.656	62.597	0.761	8.718	83.983	1.157
				SSP	75.661	67.907	77.463	65.646	67.468	0.731	8.737	88.615	1.171
03_NH	116 920	265 402	s	NS	0.825	0.817	0.828	0.803	0.839	2.227	19.292	0.878	1.064
				COS	1.552	1.530	1.511	1.477	1.391	4.139	20.656	1.440	0.953
				CAS	0.344	0.327	0.346	0.317	0.312	1.519	19.187	0.360	1.046
				SSP	**0.265**	0.248	0.270	0.241	0.249	1.489	19.219	0.296	1.117
04_NV	261 155	620 924	s	NS	2.714	2.694	2.793	2.735	2.766	4.584	45.380	2.857	1.053
				COS	5.535	5.479	5.229	5.150	4.846	8.886	49.028	4.962	0.949
				CAS	0.910	0.870	0.905	0.838	0.823	3.140	45.975	0.937	1.035
				SSP	0.766	0.726	**0.765**	0.699	0.722	3.046	45.943	0.835	1.091
05_WI	519 157	1 266 876	s	NS	10.294	10.254	10.464	10.347	10.319	9.468	96.319	10.509	1.021
				COS	14.985	14.863	14.136	13.962	13.424	19.566	102.650	13.670	0.967
				CAS	2.671	2.588	**2.593**	2.453	2.275	5.802	95.495	2.517	0.971
				SSP	3.768	3.684	3.621	3.481	3.573	5.621	97.002	3.817	1.054
06_FL	1 048 506	2 653 624	s	NS	–	–	–	–	–	–	–	–	–
				COS	–	–	–	–	–	–	–	–	–
				CAS	6.671	6.491	6.544	6.239	5.389	11.218	212.030	5.913	0.904
				SSP	4.546	4.367	**4.422**	4.116	4.223	11.238	212.130	4.748	1.074
07_TX	2 073 870	5 156 088	s	NS	–	–	–	–	–	–	–	–	–
				COS	–	–	–	–	–	–	–	–	–
				CAS	23.749	23.388	**22.782**	22.140	19.909	28.615	425.920	20.974	0.921
				SSP	25.060	24.698	23.900	23.251	23.783	26.153	423.570	24.832	1.039

Table 11: Benchmark family `vision_rnd`: Bone CT volume crop (real data). The first column lists the instances, ordered by increasing number of nodes n.

					Baselines				Verified pipeline				
					Pure		Adapted						
Instance	n	m	Unit	Alg.	Total	Init+loop	Total	Init+loop	Oracle	Cert. [ms]	Check [ms]	Total	Ratio
01_subxyz_subxy	245 762	1 431 453	s	NS	**3.609**	3.574	3.632	3.523	3.722	7.160	91.600	3.895	1.079
				COS	5.070	4.972	4.880	4.728	4.563	14.660	101.790	4.792	0.982
				CAS	40.383	40.311	39.984	39.847	39.909	9.281	92.261	40.127	1.004
				SSP	78.387	78.314	78.336	78.198	79.068	9.282	91.214	79.286	1.012
02_subxyz_subx	491 522	2 877 382	s	NS	13.484	13.413	**13.340**	13.116	13.979	15.936	183.820	14.328	1.074
				COS	13.882	13.689	13.502	13.187	12.804	30.518	204.160	13.259	0.982
				CAS	197.260	197.110	194.100	193.810	194.650	20.542	185.730	195.100	1.005
				SSP	287.370	287.210	281.490	281.200	284.930	20.454	183.050	285.380	1.014
03_subxyz	983 042	5 776 516	s	NS	103.890	103.720	104.440	103.930	104.800	36.647	371.970	105.520	1.016
				COS	43.450	43.046	**42.520**	41.844	40.598	63.089	410.010	41.513	0.976
				CAS	646.570	646.270	626.530	625.920	632.680	31.143	371.450	633.540	1.011
				SSP	1881.600	1881.300	1811.100	1810.500	1828.100	40.576	371.210	1829.000	1.010
04_subxy	1 949 698	11 520 153	s	NS	677.740	677.390	681.330	680.310	697.400	72.221	740.420	698.850	1.031
				COS	104.700	103.850	**101.730**	100.290	98.672	129.710	821.440	100.560	0.988
				CAS	–	–	–	–	–	–	–	–	–
				SSP	–	–	–	–	–	–	–	–	–
05_subx	3 899 394	23 091 149	s	NS	3978.000	3977.300	3981.100	3979.000	4105.400	136.610	1498.000	4108.300	1.033
				COS	269.490	267.800	**264.080**	261.150	253.600	246.370	1680.000	257.380	0.975
				CAS	–	–	–	–	–	–	–	–	–
				SSP	–	–	–	–	–	–	–	–	–

Table 12: Benchmark family `vision_prop`: Bone CT volume crop (real data). The first column lists the instances, ordered by increasing number of nodes n.

					Baselines				Verified pipeline				
					Pure		Adapted						
Instance	n	m	Unit	Alg.	Total	Init+loop	Total	Init+loop	Oracle	Cert. [ms]	Check [ms]	Total	Ratio
01_subxyz_subxy	245 762	1 431 453	s	NS	**3.658**	3.624	3.694	3.583	3.803	6.462	90.340	3.972	1.086
				COS	10.618	10.521	10.149	9.993	9.700	15.190	101.250	9.926	0.978
				CAS	72.486	72.414	72.295	72.158	71.919	9.342	91.121	72.137	0.998
				SSP	157.160	157.090	157.100	156.970	157.900	10.568	91.211	158.120	1.006
02_subxyz_subx	491 522	2 877 382	s	NS	13.418	13.348	**13.359**	13.135	13.758	15.930	180.940	14.104	1.056
				COS	24.923	24.721	24.123	23.799	23.194	37.510	204.170	23.666	0.981
				CAS	256.020	255.860	250.240	249.950	252.750	17.674	183.980	253.190	1.012
				SSP	676.870	676.720	655.880	655.590	671.910	20.576	183.530	672.350	1.025
03_subxyz	983 042	5 776 516	s	NS	81.132	80.964	82.418	81.911	82.262	36.453	363.390	82.971	1.023
				COS	82.801	82.387	**80.294**	79.590	78.116	73.272	410.860	79.061	0.985
				CAS	871.320	871.010	832.590	831.970	844.670	31.782	366.570	845.520	1.016
				SSP	2726.900	2726.600	2609.900	2609.300	2616.000	31.470	367.410	2616.900	1.003
04_subxy	1 949 698	11 520 153	s	NS	426.600	426.270	430.520	429.480	439.110	52.286	726.790	440.480	1.033
				COS	192.730	191.890	**187.640**	186.250	183.020	128.670	823.300	184.900	0.985
				CAS	–	–	–	–	–	–	–	–	–
				SSP	–	–	–	–	–	–	–	–	–
05_subx	3 899 394	23 091 149	s	NS	1707.300	1706.600	1750.700	1748.600	1811.200	143.780	1465.000	1814.100	1.063
				COS	455.500	453.820	**446.790**	443.840	433.850	225.940	1657.000	437.550	0.979
				CAS	–	–	–	–	–	–	–	–	–
				SSP	–	–	–	–	–	–	–	–	–

A.2 Additional Benchmarks

Table 13: Benchmark family `gridgen_8`: A node sweep at the fixed density $m = 8n$. The running parameter k of the first column is the binary logarithm of the nominal number of nodes.

					Baselines				Verified pipeline				
					Pure		Adapted						
k	n	m	Unit	Alg.	Total	Init+loop	Total	Init+loop	Oracle	Cert. [ms]	Check [ms]	Total	Ratio
08	257	2056	ms	NS	**0.461**	0.416	0.569	0.417	0.439	0.009	0.167	0.852	1.849
				COS	1.514	1.369	1.581	1.345	1.369	0.022	0.195	1.855	1.225
				CAS	3.903	3.795	4.093	3.877	3.715	0.021	0.166	4.150	1.063
				SSP	4.763	4.654	5.014	4.806	4.565	0.022	0.166	5.002	1.050
11	2071	16 568	ms	NS	**5.676**	5.318	6.625	5.367	5.413	0.040	1.508	7.797	1.374
				COS	14.831	13.652	15.538	13.592	13.532	0.209	1.663	16.720	1.127
				CAS	81.378	80.441	83.329	81.547	80.033	0.222	1.533	83.102	1.021
				SSP	153.230	152.310	156.100	154.320	151.340	0.230	1.548	154.470	1.008
12	4097	32 776	ms	NS	**15.368**	14.648	17.245	14.704	14.929	0.195	2.091	18.819	1.225
				COS	30.263	27.803	31.691	27.742	27.821	0.448	2.325	33.077	1.093
				CAS	229.890	228.000	233.630	230.100	228.270	0.373	2.086	233.290	1.015
				SSP	544.680	542.830	551.550	548.070	544.570	0.359	2.108	549.590	1.009
13	8191	65 528	s	NS	**0.043**	0.041	0.045	0.041	0.041	0.414	4.132	0.049	1.147
				COS	0.071	0.066	0.072	0.065	0.064	0.926	4.612	0.075	1.059
				CAS	0.709	0.705	0.714	0.708	0.705	0.614	4.148	0.715	1.008
				SSP	1.787	1.783	1.791	1.784	1.778	0.605	4.147	1.788	1.001
14	16 385	131 080	s	NS	**0.097**	0.094	0.105	0.095	0.094	0.856	8.249	0.110	1.131
				COS	0.129	0.117	0.134	0.116	0.114	1.753	9.210	0.135	1.049
				CAS	2.164	2.156	2.188	2.172	2.176	1.081	8.261	2.196	1.015
				SSP	4.656	4.647	4.694	4.678	4.690	1.161	8.273	4.710	1.012
15	32 762	262 096	s	NS	0.338	0.331	0.309	0.291	0.264	1.539	16.782	0.295	0.953
				COS	**0.303**	0.277	0.308	0.272	0.266	2.719	18.681	0.307	1.015
				CAS	7.036	7.014	7.048	7.016	7.085	2.026	16.793	7.125	1.013
				SSP	13.678	13.658	13.686	13.654	13.822	1.795	16.799	13.862	1.013
16	65 537	524 296	s	NS	0.789	0.778	0.809	0.765	0.769	2.668	34.580	0.831	1.053
				COS	**0.680**	0.625	0.694	0.611	0.592	5.359	38.607	0.680	0.999
				CAS	24.213	24.169	24.337	24.260	24.390	3.549	34.638	24.476	1.011
				SSP	49.802	49.758	49.769	49.696	50.636	3.571	34.660	50.722	1.019
17	131 045	1 048 360	s	NS	2.319	2.292	2.365	2.283	2.332	5.550	72.454	2.464	1.062
				COS	**1.671**	1.543	1.673	1.502	1.438	9.594	80.471	1.624	0.972
				CAS	82.473	82.367	82.252	82.096	82.089	7.106	72.671	82.278	1.000
				SSP	181.770	181.660	179.880	179.720	182.260	7.900	72.515	182.450	1.014
18	262 145	2 097 160	s	NS	6.661	6.609	6.791	6.587	6.691	10.878	153.580	6.967	1.046
				COS	**4.177**	3.875	4.213	3.795	3.504	18.751	168.600	3.913	0.937
				CAS	295.460	295.210	292.360	291.990	292.460	14.996	151.660	292.880	1.002
				SSP	580.440	580.190	573.700	573.320	583.040	14.947	152.390	583.460	1.017
19	524 177	4 193 416	s	NS	20.530	20.413	20.835	20.463	21.183	24.738	326.020	21.757	1.060
				COS	**9.938**	9.172	10.076	9.096	8.154	39.113	364.770	9.039	0.910
				CAS	875.580	874.920	893.410	892.490	937.410	29.151	324.020	938.330	1.072
				SSP	1543.000	1542.300	1561.900	1560.900	1663.700	25.354	327.750	1664.600	1.079
20	1 048 577	8 388 616	s	NS	67.860	67.580	67.849	66.878	67.808	38.889	708.110	68.981	1.017
				COS	**22.178**	20.415	22.481	20.090	18.162	75.170	786.360	20.145	0.908
				CAS	–	–	–	–	–	–	–	–	–
				SSP	–	–	–	–	–	–	–	–	–
21	2 096 705	16 773 640	s	NS	206.830	206.240	207.480	205.600	214.570	84.525	1814.000	217.340	1.051
				COS	**48.205**	44.430	49.096	44.058	40.429	146.430	2073.000	44.811	0.930
				CAS	–	–	–	–	–	–	–	–	–
				SSP	–	–	–	–	–	–	–	–	–
22	4 194 305	33 554 440	s	NS	1031.600	1030.000	1031.100	1025.400	1078.000	166.610	5616.000	1085.600	1.053
				COS	**114.270**	106.160	116.210	104.190	96.518	259.350	6345.000	108.070	0.946
				CAS	–	–	–	–	–	–	–	–	–
				SSP	–	–	–	–	–	–	–	–	–

Table 14: Benchmark family `gridgen_sr`: A node sweep at the density $m = n^{1.5}$. The running parameter k of the first column is the binary logarithm of the nominal number of nodes.

					Baselines				Verified pipeline				
					Pure		Adapted						
k	n	m	Unit	Alg.	Total	Init+loop	Total	Init+loop	Oracle	Cert. [ms]	Check [ms]	Total	Ratio
08	257	4112	ms	NS	**0.675**	0.589	0.859	0.576	0.609	0.012	0.288	1.195	1.771
				COS	2.110	1.839	2.286	1.840	1.889	0.023	0.329	2.632	1.247
				CAS	7.194	6.982	7.637	7.229	6.781	0.023	0.284	7.477	1.039
				SSP	6.676	6.463	7.108	6.710	6.224	0.024	0.288	6.918	1.036
09	507	11 661	ms	NS	**1.804**	1.529	2.470	1.692	1.559	0.014	1.026	3.193	1.770
				COS	5.076	4.275	5.576	4.322	4.476	0.071	1.151	6.584	1.297
				CAS	29.358	28.745	30.758	29.616	27.724	0.045	1.028	29.697	1.012
				SSP	26.479	25.855	27.755	26.644	24.938	0.044	1.029	26.933	1.017
10	1025	32 800	ms	NS	**5.567**	4.942	7.390	4.959	5.059	0.171	2.847	9.589	1.723
				COS	14.982	12.714	16.752	12.977	13.269	0.344	3.161	19.259	1.286
				CAS	110.380	108.620	114.520	110.980	104.550	0.407	2.838	110.180	0.998
				SSP	130.750	128.990	137.090	133.560	126.090	0.399	2.830	131.730	1.008
11	2071	93 195	ms	NS	**15.902**	14.005	20.285	14.052	14.358	0.388	7.981	26.690	1.678
				COS	42.513	34.993	47.583	35.989	36.526	0.962	8.713	52.721	1.240
				CAS	449.440	443.340	459.520	449.020	428.080	0.864	7.912	443.440	0.987
				SSP	449.930	443.660	464.910	454.180	436.890	0.886	7.943	452.600	1.006
12	4097	262 208	s	NS	**0.053**	0.047	0.068	0.047	0.048	1.155	23.129	0.084	1.577
				COS	0.128	0.102	0.147	0.106	0.108	2.603	25.272	0.156	1.219
				CAS	2.263	2.242	2.290	2.253	2.181	1.950	23.109	2.226	0.984
				SSP	1.984	1.964	2.034	1.996	1.959	1.811	22.926	2.003	1.009
13	8191	745 381	s	NS	**0.193**	0.176	0.233	0.176	0.176	2.843	44.041	0.258	1.336
				COS	0.438	0.340	0.476	0.350	0.354	6.087	50.770	0.476	1.087
				CAS	9.397	9.318	9.408	9.290	9.031	5.320	44.121	9.152	0.974
				SSP	8.239	8.159	8.421	8.302	8.137	4.650	44.086	8.255	1.002
14	16 385	2 097 280	s	NS	**0.616**	0.562	0.751	0.562	0.567	7.676	122.620	0.803	1.303
				COS	2.062	1.598	2.210	1.624	1.409	16.186	142.250	1.791	0.869
				CAS	34.908	34.481	34.804	34.253	33.378	14.850	122.960	33.763	0.970
				SSP	33.964	33.539	34.982	34.432	33.588	15.831	122.870	33.976	1.000
15	32 762	5 929 922	s	NS	**2.695**	2.512	2.895	2.348	2.161	20.694	357.910	2.838	1.053
				COS	7.173	5.356	7.660	5.463	4.462	45.552	412.590	5.608	0.782
				CAS	168.460	166.750	165.250	163.210	163.040	42.852	358.800	164.210	0.994
				SSP	133.370	131.670	135.670	133.600	132.450	36.861	359.840	133.600	1.002
16	65 537	16 777 472	s	NS	**8.538**	7.786	9.703	7.646	7.773	64.201	1115.000	10.003	1.172
				COS	21.810	15.955	23.549	16.225	13.184	124.260	1282.000	16.651	0.763
				CAS	1043.400	1037.900	1083.400	1076.300	1081.200	85.833	1110.000	1084.600	1.040
				SSP	524.510	519.050	554.960	547.820	565.820	104.790	1114.000	569.340	1.085

Table 15: Benchmark family `gridgen_deg`: A density sweep at the fixed node count $n = 4097$. The running parameter i of the first column is the binary logarithm of the average degree m/n.

					Baselines				Verified pipeline				
					Pure		Adapted						
i	n	m	Unit	Alg.	Total	Init+loop	Total	Init+loop	Oracle	Cert. [ms]	Check [ms]	Total	Ratio
02	4097	16 388	ms	NS	**5.980**	5.553	6.865	5.533	5.700	0.042	1.592	8.298	1.388
				COS	14.870	13.729	15.451	13.607	12.989	0.227	1.719	16.326	1.098
				CAS	66.130	65.268	66.259	64.645	65.121	0.152	1.623	68.331	1.033
				SSP	10.742	9.920	11.444	9.883	9.860	0.101	1.594	12.935	1.204
03	4097	32 776	ms	NS	**15.377**	14.656	17.272	14.792	14.904	0.196	2.082	18.778	1.221
				COS	30.287	27.811	31.603	27.678	27.611	0.439	2.326	32.884	1.086
				CAS	230.260	228.340	233.820	230.270	228.270	0.385	2.082	233.300	1.013
				SSP	546.460	544.570	551.280	547.690	543.860	0.363	2.095	548.920	1.005
04	4097	65 552	s	NS	**0.023**	0.022	0.027	0.022	0.023	0.321	3.971	0.030	1.276
				COS	0.043	0.038	0.047	0.038	0.038	0.810	4.526	0.048	1.115
				CAS	0.426	0.422	0.438	0.430	0.420	0.661	3.975	0.430	1.007
				SSP	1.021	1.017	1.050	1.043	1.017	0.694	3.963	1.026	1.006
05	4097	131 104	s	NS	**0.032**	0.029	0.039	0.029	0.030	0.621	11.495	0.048	1.502
				COS	0.070	0.059	0.078	0.060	0.060	1.462	12.576	0.083	1.187
				CAS	0.829	0.820	0.848	0.831	0.812	1.123	11.315	0.835	1.007
				SSP	1.209	1.200	1.241	1.225	1.208	1.088	11.355	1.230	1.017
06	4097	262 208	s	NS	**0.053**	0.047	0.068	0.048	0.048	1.200	23.182	0.084	1.580
				COS	0.128	0.102	0.147	0.105	0.108	2.704	25.338	0.155	1.213
				CAS	2.261	2.240	2.288	2.250	2.182	1.934	23.167	2.227	0.985
				SSP	1.988	1.967	2.033	1.995	1.964	1.943	23.078	2.010	1.011
07	4097	524 416	s	NS	**0.085**	0.073	0.115	0.073	0.074	2.205	45.400	0.146	1.715
				COS	0.255	0.192	0.285	0.199	0.204	4.469	49.801	0.299	1.174
				CAS	6.403	6.349	6.365	6.282	6.048	3.472	45.614	6.140	0.965
				SSP	3.031	2.980	3.133	3.049	2.947	3.153	45.501	3.038	1.003
08	4097	1 048 832	s	NS	**0.132**	0.106	0.198	0.109	0.107	3.927	60.637	0.252	1.911
				COS	0.599	0.449	0.670	0.464	0.459	8.468	70.253	0.665	1.109
				CAS	17.806	17.677	17.439	17.244	16.650	6.448	60.611	16.853	0.966
				SSP	4.787	4.657	5.031	4.837	4.574	6.483	60.599	4.780	0.999
09	4097	2 097 664	s	NS	**0.238**	0.175	0.373	0.174	0.174	8.586	181.150	0.472	1.982
				COS	1.785	1.321	1.939	1.353	1.149	16.499	198.750	1.585	0.888
				CAS	126.340	125.920	122.160	121.600	115.550	13.532	179.810	115.980	0.949
				SSP	7.041	6.612	7.492	6.921	6.478	13.757	183.330	6.920	0.983
10	4097	4 195 328	s	NS	**0.500**	0.311	0.795	0.311	0.316	16.629	364.520	0.930	1.860
				COS	4.235	3.068	4.575	3.130	2.507	32.575	400.640	3.421	0.808
				CAS	549.100	548.000	528.320	526.960	501.030	27.613	360.990	501.930	0.950
				SSP	13.634	12.544	14.644	13.277	12.384	25.303	360.760	13.281	0.974
11	4097	8 390 656	s	NS	**0.929**	0.543	1.538	0.542	0.546	29.081	727.780	1.741	1.874
				COS	9.567	6.924	10.294	7.066	5.715	56.074	797.350	7.545	0.789
				CAS	1569.400	1566.900	1500.300	1497.300	1424.300	60.493	734.250	1426.100	0.951
				SSP	27.261	24.779	29.330	26.304	24.552	52.005	724.070	26.369	0.967
12	4097	16 781 312	s	NS	**1.714**	0.912	3.006	0.933	0.957	59.228	1471.000	3.377	1.971
				COS	19.941	14.187	21.445	14.444	11.638	127.000	1614.000	15.367	0.771
				CAS	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF	DNF
				SSP	49.450	44.165	53.750	47.351	43.839	107.050	1455.000	47.539	0.961

Table 16: Benchmark family `netgen_lo_8`: A node sweep at the fixed density $m = 8n$. The running parameter k of the first column is the binary logarithm of the nominal number of nodes.

					Baselines				Verified pipeline				
					Pure		Adapted						
k	n	m	Unit	Alg.	Total	Init+loop	Total	Init+loop	Oracle	Cert. [ms]	Check [ms]	Total	Ratio
08	256	2048	ms	NS	**0.224**	0.168	0.301	0.164	0.179	0.009	0.160	0.557	2.484
				COS	0.771	0.626	0.856	0.635	0.631	0.021	0.189	1.104	1.432
				CAS	0.597	0.486	0.681	0.494	0.472	0.017	0.160	0.909	1.522
				SSP	0.663	0.550	0.766	0.577	0.552	0.017	0.166	0.986	1.489
11	2048	16 384	ms	NS	**2.672**	2.273	3.290	2.264	2.348	0.035	1.514	4.729	1.770
				COS	7.288	6.033	7.800	6.021	5.958	0.190	1.640	9.065	1.244
				CAS	9.484	8.512	10.069	8.490	8.392	0.114	1.513	11.308	1.192
				SSP	16.769	15.800	17.370	15.825	15.571	0.122	1.506	18.549	1.106
12	4096	32 768	ms	NS	**6.652**	5.869	8.040	5.895	6.071	0.172	2.096	10.760	1.618
				COS	15.669	13.111	16.737	13.155	12.838	0.392	2.338	18.922	1.208
				CAS	24.470	22.450	25.642	22.441	22.428	0.251	2.104	28.231	1.154
				SSP	48.205	46.213	49.652	46.378	46.271	0.308	2.087	52.144	1.082
13	8192	65 536	ms	NS	**15.334**	13.815	18.254	13.895	14.134	0.312	4.149	21.664	1.413
				COS	35.805	30.580	38.189	30.814	29.841	0.724	4.650	40.014	1.118
				CAS	66.976	62.728	69.405	62.709	62.790	0.579	4.169	72.865	1.088
				SSP	130.570	126.320	133.410	126.670	126.970	0.637	4.200	137.060	1.050
14	16 384	131 072	ms	NS	**41.104**	38.061	47.106	38.106	38.484	0.774	8.434	53.910	1.312
				COS	78.984	67.658	84.280	67.966	66.591	1.608	9.327	87.771	1.111
				CAS	192.660	183.450	198.280	183.610	184.760	1.104	8.329	205.400	1.066
				SSP	390.100	381.310	397.020	382.210	384.540	0.996	8.328	404.670	1.037
15	32 768	262 144	s	NS	**0.119**	0.113	0.130	0.111	0.113	1.448	17.098	0.144	1.208
				COS	0.175	0.148	0.185	0.149	0.144	2.803	19.044	0.188	1.075
				CAS	0.448	0.428	0.461	0.429	0.433	1.875	17.107	0.475	1.060
				SSP	1.168	1.147	1.178	1.146	1.153	1.714	17.061	1.194	1.022
16	65 536	524 288	s	NS	**0.317**	0.305	0.343	0.303	0.308	2.686	36.163	0.372	1.171
				COS	0.408	0.345	0.420	0.341	0.332	5.367	40.232	0.423	1.037
				CAS	1.456	1.407	1.470	1.399	1.415	3.066	36.107	1.502	1.032
				SSP	3.719	3.670	3.699	3.628	3.691	3.613	36.146	3.780	1.022
17	131 072	1 048 576	s	NS	**0.935**	0.910	0.987	0.903	0.935	5.246	76.426	1.069	1.143
				COS	1.003	0.861	1.032	0.850	0.818	9.615	85.056	1.013	1.009
				CAS	4.188	4.070	4.208	4.046	4.086	5.985	76.082	4.278	1.022
				SSP	9.729	9.614	9.693	9.531	9.618	6.412	76.456	9.812	1.012
18	262 144	2 097 152	s	NS	2.857	2.805	2.969	2.789	2.925	9.710	165.120	3.207	1.122
				COS	**2.744**	2.377	2.830	2.368	2.168	18.450	179.010	2.589	0.944
				CAS	11.641	11.330	11.676	11.262	11.511	12.534	162.590	11.939	1.026
				SSP	32.145	31.826	32.249	31.829	32.251	14.990	161.930	32.687	1.017
19	524 288	4 194 304	s	NS	8.921	8.808	9.276	8.881	9.398	19.878	353.410	9.986	1.119
				COS	**6.717**	5.836	7.074	5.937	5.342	37.592	389.920	6.266	0.933
				CAS	35.708	34.920	36.276	35.214	38.449	29.310	350.100	39.416	1.104
				SSP	97.104	96.325	99.473	98.402	104.990	25.225	353.240	105.940	1.091
20	1 048 576	8 388 608	s	NS	27.958	27.693	29.313	28.398	29.808	39.223	769.170	31.039	1.110
				COS	**15.874**	13.881	16.392	13.773	12.842	75.132	872.630	14.873	0.937
				CAS	–	–	–	–	–	–	–	–	–
				SSP	–	–	–	–	–	–	–	–	–
21	2 097 152	16 777 216	s	NS	104.760	104.120	106.830	104.350	101.620	96.743	2234.000	104.850	1.001
				COS	**37.901**	33.727	39.793	33.751	31.852	147.730	2534.000	36.795	0.971
				CAS	–	–	–	–	–	–	–	–	–
				SSP	–	–	–	–	–	–	–	–	–
22	4 194 304	33 554 432	s	NS	374.640	372.980	385.210	378.750	420.320	139.260	6959.000	429.170	1.146
				COS	**93.192**	84.132	96.176	82.626	79.457	273.520	8002.000	92.658	0.994
				CAS	–	–	–	–	–	–	–	–	–
				SSP	–	–	–	–	–	–	–	–	–

Table 17: Benchmark family `netgen_lo_sr`: A node sweep at the density $m = n^{1.5}$. The running parameter k of the first column is the binary logarithm of the nominal number of nodes.

					Baselines				Verified pipeline				
					Pure		Adapted						
k	n	m	Unit	Alg.	Total	Init+loop	Total	Init+loop	Oracle	Cert. [ms]	Check [ms]	Total	Ratio
08	256	4096	ms	NS	**0.333**	0.243	0.490	0.238	0.266	0.010	0.279	0.842	2.526
				COS	1.181	0.909	1.328	0.924	0.936	0.022	0.324	1.675	1.418
				CAS	0.847	0.642	1.036	0.670	0.635	0.017	0.279	1.336	1.577
				SSP	1.007	0.804	1.182	0.821	0.792	0.017	0.284	1.491	1.480
09	512	11 585	ms	NS	**0.919**	0.654	1.443	0.648	0.696	0.015	1.040	2.327	2.533
				COS	3.275	2.498	3.757	2.541	2.597	0.069	1.163	4.722	1.442
				CAS	2.750	2.172	3.300	2.222	2.093	0.025	1.043	4.064	1.478
				SSP	3.813	3.233	4.391	3.307	3.108	0.025	1.031	5.059	1.327
10	1024	32 768	ms	NS	**2.491**	1.800	3.830	1.793	1.871	0.163	2.837	6.367	2.556
				COS	8.887	6.687	10.144	6.845	7.129	0.337	3.144	12.959	1.458
				CAS	8.830	7.117	10.262	7.290	7.014	0.244	2.859	12.537	1.420
				SSP	12.275	10.520	13.701	10.701	10.258	0.248	2.820	15.696	1.279
11	2048	92 682	ms	NS	**7.642**	5.785	11.959	5.801	6.210	0.352	7.849	18.250	2.388
				COS	26.522	19.930	31.293	20.731	20.397	0.900	8.700	36.424	1.373
				CAS	33.027	27.669	37.892	28.135	27.678	0.702	7.888	42.977	1.301
				SSP	45.301	39.974	50.293	40.713	39.951	0.715	7.945	55.287	1.220
12	4096	262 144	ms	NS	**24.923**	18.225	36.587	18.424	17.844	1.015	22.485	52.895	2.122
				COS	83.160	60.812	95.423	62.898	62.470	2.652	24.999	108.950	1.310
				CAS	104.510	86.133	117.130	86.604	86.544	1.938	22.628	130.160	1.245
				SSP	161.690	143.710	174.960	145.430	144.580	1.916	22.505	188.000	1.163
13	8192	741 455	ms	NS	**74.884**	57.349	111.550	57.279	58.671	3.061	43.118	139.510	1.863
				COS	278.100	203.740	317.370	211.180	203.060	6.084	50.223	316.650	1.139
				CAS	333.960	272.490	373.590	277.110	278.370	4.539	43.098	385.550	1.154
				SSP	628.950	561.170	670.070	569.580	562.450	4.815	43.155	670.990	1.067
14	16 384	2 097 152	s	NS	**0.266**	0.208	0.371	0.208	0.203	8.519	122.630	0.450	1.692
				COS	1.339	1.030	1.440	1.041	0.822	16.678	142.450	1.168	0.872
				CAS	1.328	1.060	1.437	1.074	1.081	11.431	124.070	1.411	1.062
				SSP	2.366	2.092	2.492	2.122	2.104	12.878	121.980	2.437	1.030
15	32 768	5 931 642	s	NS	**0.908**	0.727	1.261	0.718	0.754	20.810	354.350	1.427	1.572
				COS	4.967	3.827	5.414	3.909	2.832	44.251	409.170	3.851	0.775
				CAS	4.485	3.448	4.866	3.494	3.539	30.940	353.300	4.520	1.008
				SSP	8.241	7.205	8.714	7.340	7.417	33.568	352.800	8.397	1.019
16	65 536	16 777 216	s	NS	**3.314**	2.584	4.425	2.609	2.822	54.006	1102.000	5.080	1.533
				COS	16.159	12.618	17.548	12.790	9.482	113.520	1267.000	12.594	0.779
				CAS	14.883	11.604	16.977	12.193	12.679	104.290	1106.000	15.775	1.060
				SSP	28.816	25.502	31.811	26.988	27.502	109.080	1106.000	30.616	1.062

Table 18: Benchmark family `vision_inv`: Bone CT volume crop (real data). The first column lists the instances, ordered by increasing number of nodes n.

					Baselines				Verified pipeline				
					Pure		Adapted						
Instance	n	m	Unit	Alg.	Total	Init+loop	Total	Init+loop	Oracle	Cert. [ms]	Check [ms]	Total	Ratio
01_subxyz_subxy	245 762	1 431 453	s	NS	**2.582**	2.548	2.640	2.530	2.608	7.290	89.288	2.779	1.076
				COS	8.630	8.535	8.244	8.089	7.870	16.132	101.010	8.102	0.983
				CAS	11.963	11.890	11.994	11.856	11.811	8.740	91.819	12.028	1.005
				SSP	30.452	30.380	30.480	30.341	30.740	9.346	90.059	30.957	1.017
02_subxyz_subx	491 522	2 877 382	s	NS	10.645	10.574	**10.640**	10.416	10.668	15.300	179.580	11.012	1.035
				COS	21.374	21.179	20.705	20.391	19.818	31.642	205.490	20.276	0.979
				CAS	52.677	52.521	51.458	51.168	51.560	20.843	185.130	52.005	1.011
				SSP	114.860	114.700	113.240	112.950	114.980	17.842	180.650	115.410	1.019
03_subxyz	983 042	5 776 516	s	NS	52.972	52.804	**52.614**	52.114	53.534	31.641	360.850	54.223	1.031
				COS	66.708	66.303	64.888	64.182	62.926	63.036	415.450	63.848	0.984
				CAS	220.900	220.600	212.690	212.110	212.810	40.204	372.990	213.700	1.005
				SSP	834.840	834.540	804.290	803.700	803.150	36.191	365.150	804.020	1.000
04_subxy	1 949 698	11 520 153	s	NS	381.930	381.590	372.180	371.200	389.060	62.368	717.160	390.450	1.049
				COS	164.760	163.940	**160.460**	159.020	156.340	128.750	826.170	158.230	0.986
				CAS	–	–	–	–	–	–	–	–	–
				SSP	–	–	–	–	–	–	–	–	–
05_subx	3 899 394	23 091 149	s	NS	1979.000	1978.300	1984.100	1981.900	2108.700	111.750	1454.000	2111.500	1.067
				COS	409.280	407.660	**401.780**	398.900	391.180	216.180	1641.000	394.840	0.983
				CAS	–	–	–	–	–	–	–	–	–
				SSP	–	–	–	–	–	–	–	–	–

A.3 Fitting the Cost Model

Table 19 lists, per benchmark family and algorithm, the parameters of the cost model

$$\text{ratio} \;\approx\; \beta + \frac{144\,\text{ns}\cdot m + 71\,\text{ns}\cdot n}{T}$$

from the evaluation, together with the error the model leaves on the 617 rows of the preceding tables in which the solver and at least one baseline finished. Here T is the better baseline's compute time, i.e. the denominator of the *Ratio* column, and the two constants are the same for all groups. For each of the 171 boxed rows we divided the solver's compute time minus the oracle's by m; over the 18 families, the medians of this cost per arc lie on a straight line in n/m, and a least-squares fit of that line gives the two constants.

Within one group, β is the median over the k instances of the oracle's compute time divided by T, with the smallest and largest value of that quotient in the next two columns; it is measured, not fitted. For T we fit a power law $T_{\text{fit}} = c\,m^{\alpha}$ to the instances of the group by ordinary least squares on $\log T$ and $\log m$; the table lists the exponent α and δ_T, the median over the instances of $|T_{\text{fit}} - T|/T$, so T_{fit} is the baseline time that this power law predicts from m alone.

The error ε of a row is the absolute difference between the ratio the model predicts and the measured ratio; the two ε columns are its median over the k instances of the group, and the column $\overline{\text{ratio}}$ gives the mean measured ratio for comparison; it is close to 1 in most groups and exceeds 1.3 only in six NS groups. The two ε columns differ in what is inserted for T: ε (meas.) uses the row's measured baseline time and thus tests the model with β as its only parameter, whereas ε (fit) inserts T_{fit} and thus tests a prediction from n and m alone, as for an instance that has not been run. Over all 617 rows the median misprediction is 0.023 with the measured T and 0.022 with T_{fit}; 90 % of the rows are off by less than 0.12 and 0.14, respectively. The larger errors sit in the 76 rows with T below 10 ms (median 0.067), where noise and start-up costs dominate; on the other 541 rows the median is 0.019, so the model captures the development with growing size, its actual purpose, to about two hundredths.

Table 19: Parameters of the cost model per benchmark family and algorithm. k is the number of instances with a finished run; β is the median of the oracle's compute time over the better baseline's compute time, with the smallest and largest value of that quotient next to it; α is the exponent of the fitted baseline time $T \sim m^{\alpha}$ and δ_T the median relative deviation of this fit from the measured times; $\overline{\text{ratio}}$ is the mean of the measured ratios of the group, and the two ε columns give the median absolute error between model and measured ratio, once with the measured T and once with the fitted T.

Family	Alg.	k	β	$\beta_{\min}$	$\beta_{\max}$	α	δ_T [%]	$\overline{\text{ratio}}$	ε (meas.)	ε (fit)
`netgen_8`	NS	13	0.963	0.925	1.027	1.56	15	1.120	0.025	0.021
	COS	13	0.876	0.835	0.900	1.20	8	1.030	0.026	0.038
	CAS	7	1.003	0.958	1.012	1.98	14	1.003	0.008	0.008
	SSP	10	1.005	0.939	1.090	1.74	6	1.021	0.008	0.008
`netgen_sr`	NS	9	0.911	0.887	0.980	1.18	13	1.485	0.057	0.054
	COS	9	0.834	0.647	0.890	1.13	17	1.124	0.124	0.145
	CAS	7	0.956	0.948	0.982	1.78	19	0.969	0.005	0.009
	SSP	9	0.982	0.925	1.063	1.38	3	1.021	0.019	0.018
`netgen_deg`	NS	12	0.873	0.587	1.000	0.70	15	1.588	0.081	0.087
	COS	12	0.823	0.638	0.906	0.93	38	1.052	0.098	0.153
	CAS	9	0.978	0.953	1.004	1.64	9	0.982	0.022	0.022
	SSP	12	0.978	0.937	1.010	0.76	17	1.005	0.019	0.017

Table 19: Parameters of the cost model (continued).

Family	Alg.	k	β	$\beta_{\min}$	$\beta_{\max}$	α	δ_T [%]	$\overline{\text{ratio}}$	ε (meas.)	ε (fit)
goto_8	NS	10	0.971	0.954	1.042	2.13	56	1.067	0.023	0.026
	COS	11	0.960	0.942	1.019	1.43	21	1.003	0.012	0.015
	CAS	9	0.973	0.928	0.997	1.61	68	0.982	0.015	0.020
	SSP	7	0.966	0.949	0.994	2.09	8	0.973	0.007	0.008
goto_sr	NS	9	0.988	0.879	1.002	1.64	23	1.076	0.028	0.036
	COS	9	1.032	0.970	1.111	1.23	8	1.105	0.027	0.032
	CAS	7	0.972	0.957	1.000	1.85	7	0.981	0.007	0.007
	SSP	5	0.941	0.919	0.957	2.24	6	0.942	0.008	0.008
grid_wide	NS	11	0.987	0.901	1.007	1.23	14	1.297	0.025	0.015
	COS	11	0.899	0.791	0.934	1.47	21	0.939	0.032	0.035
	CAS	10	1.024	0.920	1.088	2.01	14	1.074	0.035	0.030
	SSP	9	1.031	0.960	1.071	2.09	6	1.061	0.020	0.019
grid_square	NS	10	0.986	0.648	1.032	1.80	37	1.143	0.013	0.036
	COS	11	0.939	0.887	1.005	1.39	11	0.999	0.011	0.012
	CAS	11	0.969	0.902	0.981	1.63	9	1.024	0.011	0.012
	SSP	10	1.005	0.969	1.014	2.00	6	1.034	0.004	0.004
grid_long	NS	10	0.997	0.911	1.004	1.76	27	1.170	0.011	0.008
	COS	10	0.900	0.882	0.920	1.11	4	0.999	0.015	0.013
	CAS	11	0.923	0.902	0.979	0.94	11	1.174	0.023	0.011
	SSP	11	0.955	0.898	0.980	0.82	26	1.132	0.029	0.022
road_paths	NS	5	1.003	0.968	1.019	1.71	5	1.098	0.010	0.011
	COS	5	0.907	0.877	0.943	1.51	4	0.977	0.015	0.012
	CAS	7	0.874	0.736	0.937	1.37	17	1.259	0.062	0.040
	SSP	7	0.872	0.742	0.937	1.37	17	1.267	0.061	0.037
road_flow	NS	5	1.002	0.932	1.019	1.72	8	1.086	0.011	0.007
	COS	5	0.921	0.889	0.950	1.52	5	0.966	0.008	0.007
	CAS	7	0.874	0.824	0.909	1.45	12	1.043	0.016	0.029
	SSP	7	0.944	0.889	0.995	1.40	21	1.108	0.048	0.029
vision_rnd	NS	5	1.031	1.009	1.048	2.58	6	1.047	0.009	0.017
	COS	5	0.955	0.935	0.970	1.44	4	0.981	0.007	0.007
	CAS	3	1.003	0.998	1.010	1.97	7	1.007	0.004	0.004
	SSP	3	1.009	1.009	1.012	2.25	9	1.012	0.000	0.001
vision_prop	NS	5	1.030	1.014	1.061	2.27	9	1.052	0.007	0.011
	COS	5	0.971	0.956	0.975	1.38	7	0.982	0.004	0.004
	CAS	3	1.010	0.995	1.015	1.75	1	1.009	0.005	0.005
	SSP	3	1.005	1.002	1.024	2.01	1	1.011	0.002	0.002
gridgen_8	NS	13	0.974	0.853	1.045	1.50	18	1.155	0.055	0.048
	COS	13	0.870	0.819	0.919	1.17	8	1.013	0.023	0.031
	CAS	10	0.999	0.952	1.071	1.64	15	1.022	0.007	0.008
	SSP	10	1.009	0.958	1.078	1.67	9	1.022	0.009	0.009
gridgen_sr	NS	9	0.906	0.802	0.921	1.15	13	1.487	0.082	0.066
	COS	9	0.846	0.604	0.895	1.14	18	1.088	0.114	0.141
	CAS	9	0.959	0.943	1.036	1.41	10	1.000	0.005	0.008
	SSP	9	0.987	0.932	1.079	1.36	4	1.018	0.012	0.010

Table 19: Parameters of the cost model (continued).

Family	Alg.	k	β	$\beta_{\min}$	$\beta_{\max}$	α	δ_T [%]	$\overline{\text{ratio}}$	ε (meas.)	ε (fit)
`gridgen_deg`	NS	11	0.870	0.559	0.969	0.77	15	1.662	0.075	0.089
	COS	11	0.802	0.584	0.912	1.06	30	1.022	0.114	0.162
	CAS	10	0.960	0.946	0.991	1.60	41	0.983	0.012	0.013
	SSP	11	0.956	0.887	0.999	0.92	28	1.012	0.033	0.025
`netgen_lo_8`	NS	13	0.970	0.799	1.122	1.49	28	1.355	0.097	0.069
	COS	13	0.818	0.790	0.853	1.22	10	1.078	0.038	0.056
	CAS	10	0.963	0.791	1.077	1.45	7	1.127	0.023	0.022
	SSP	10	0.986	0.833	1.081	1.56	4	1.093	0.014	0.010
`netgen_lo_sr`	NS	9	0.783	0.716	0.852	1.11	10	2.087	0.154	0.158
	COS	9	0.751	0.570	0.802	1.16	17	1.174	0.122	0.149
	CAS	9	0.814	0.750	0.852	1.18	4	1.256	0.045	0.050
	SSP	9	0.889	0.786	0.954	1.24	4	1.183	0.013	0.024
`vision_inv`	NS	5	1.017	1.003	1.066	2.42	12	1.052	0.027	0.026
	COS	5	0.970	0.955	0.974	1.41	5	0.983	0.005	0.005
	CAS	3	1.001	0.987	1.002	2.06	1	1.007	0.002	0.002
	SSP	3	1.009	0.999	1.015	2.35	10	1.012	0.006	0.006

B Heuristics

One table per benchmark family and algorithm reports the effect of each heuristic of Table 20; the algorithms are the two fastest of Appendix A for that family, `netgen_deg` was not measured, and every number is a median over the instances a to e. The baseline is our solver with default settings; times are the CPU time of initialisation and main loop, so the cost of presolving, which runs before the C++ core, is not included. Each heuristic run is bracketed by two baseline runs and divided by their mean, which cancels machine drift of over 20 % within a day. A cell shows the median heuristic time, the median bracketing baseline time (*Calib.*), and the median of the five per-instance quotients (*Ratio*); the ratio is thus not the quotient of the two times beside it.

A star marks a ratio below 1 where at least four of the five instances were faster by more than the noise of the two baseline runs; the smallest starred ratio of a size is boxed. The row *stack* switches on all starred heuristics of that size together and is not measured where fewer than two helped. Below the second rule are the heuristics that never helped, with a dagger where they made the solver slower by the same criterion. Rows named *no ...* switch a default setting off; – means not measured, DNF not finished within two hours, CAP stopped at 1.8 times the baseline time.

Table 20: The heuristics of Appendix B, by the names used in the tables. The second column lists the algorithms the heuristic was tried on.

Name	Alg.	Setting
discharge	COS	after a partial augmentation the active node is not abandoned but further paths are pushed out of it until its excess is gone, i.e. the discharge operation of push-relabel [25] applied to partial augment-relabel; this is the blocking augment-relabel of the main text
highest excess	COS	discharge, but the next active node is the one with the largest excess instead of the next one in FIFO order [25]
$\alpha = 12$, $\alpha = 20$	COS	the factor by which the error bound ε is divided after each scaling phase [26]; LEMON uses 16, so 12 and 20 are one step either side of it
arc fixing	COS	speculative arc fixing [23]: arcs whose reduced cost is far enough from zero are removed from the scan
arc fixing + renumb.	COS	arc fixing, and the arcs that remain are periodically renumbered into contiguous memory
path length $\ell = 2$, path length $\ell = 8$	COS	partial augment-relabel [24] builds admissible paths of at most ℓ arcs and pushes along them; LEMON fixes $\ell = 4$
cost GCD	COS	all costs are divided by their greatest common divisor before solving
anti-stalling	NS	one of our own entering-arc rules, switched on as soon as one degenerate pivot has occurred, in place of the block search
no arc mixing	NS	LEMON interleaves the arc array so that the block search samples arcs spread over the graph; this switches the interleaving off, and the block search scans the arcs in input order
renumbering	all	the periodic vertex reordering of the implementation section: every so many iterations the nodes are renumbered in DFS order of the current spanning tree (NS) or of the search tree (COS, CAS, SSP), for cache locality
factor 2, factor 8	CAS	the factor by which the scaling parameter Δ is divided after each phase [20]; LEMON uses 4

Table 20: The heuristics of Appendix B (continued).

Name	Alg.	Setting
radix heap	CAS, SSP	a radix heap as the priority queue of the shortest-path searches instead of LEMON's binary heap
hot queue	CAS, SSP	a hot queue [12] as the priority queue of the shortest-path searches
multi-source Dijkstra	CAS, SSP	one shortest-path search from all nodes with excess at once instead of one search per source
blocking flow	CAS, SSP	after the search settles everything reachable, a blocking flow in the sense of Dinic [19] is pushed over the tight arcs; it extends multi-source Dijkstra and is measured against that setting, not against the baseline
label reuse	CAS, SSP	the distance labels of one phase are carried into the next phase as a starting potential instead of being recomputed
node presolve	all	the node presolving of the implementation section, which contracts zero-supply nodes of degree at most two before the C++ core is entered
arc presolve	all	the arc presolving of the implementation section, which removes zero-capacity arcs and merges parallel arcs of equal cost before the C++ core is entered
reverse nodes	COS	the nodes are numbered in reverse input order; the COS baseline does not reverse the nodes
reverse core arcs	COS	the arcs of the residual graph in CSR format are numbered in reverse order, in addition to the arc reversal of the baseline
no node reversal	NS, CAS, SSP	the nodes keep their input order; the baselines of these algorithms number them in reverse
no arc reversal	all	the arcs keep their input order; all four baselines number them in reverse
stack	all	every heuristic that helped on its own at this size, switched on together and measured in a bracket of its own; where two of them set the same parameter to different values, the faster one is kept

B.1 Core Benchmarks

Table 21: Heuristics for benchmark family `netgen_8` under NS; the sizes are those of Table 1.

	$k = 13$ [ms] $n = 8192$ $m = 65\,536$			$k = 16$ [s] $n = 65\,536$ $m = 524\,288$			$k = 19$ [s] $n = 524\,288$ $m = 4\,194\,304$			$k = 21$ [s] $n = 2\,097\,152$ $m = 16\,777\,216$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
renumbering	43.8	0.941*	45.7	1.003	0.813*	1.240	47.687	1.323	34.835	679.802	2.065	337.474
no node reversal	45.8	1.002	45.7	1.129	0.975	1.240	36.070	1.006	34.835	309.593	0.951*	337.474
Never helpful at any size (ratio only):												
node presolve		1.001			1.025			1.006			1.090	
arc presolve		0.989			0.998			1.013			1.018	
anti-stalling		25.503†			109.350†			DNF			DNF	
no arc reversal		0.961			0.984			1.019			0.961	
no arc mixing		1.046			1.005			1.005			0.978	

Table 22: Heuristics for benchmark family `netgen_8` under COS; the sizes are those of Table 1.

	$k = 13$ [ms] $n = 8192$ $m = 65\,536$			$k = 16$ [ms] $n = 65\,536$ $m = 524\,288$			$k = 19$ [s] $n = 524\,288$ $m = 4\,194\,304$			$k = 21$ [s] $n = 2\,097\,152$ $m = 16\,777\,216$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
discharge	58.9	1.000	58.5	649.0	1.021	635.5	9.512	1.012	9.582	54.889	0.926*	54.897
$\alpha = 20$	56.2	0.925*	58.5	560.0	0.876*	635.5	8.987	0.938	9.582	55.947	1.019	54.897
$\alpha = 12$	57.5	1.024	58.5	551.4	0.844*	635.5	9.159	0.955	9.582	59.587	0.872	54.897
reverse nodes	58.1	0.994	58.5	621.8	0.941*	635.5	9.101	0.913*	9.582	60.200	1.035	54.897
reverse core arcs	58.8	0.976*	58.5	618.1	0.954*	635.5	8.667	0.922*	9.582	58.491	1.013	54.897
path length $\ell = 8$	52.0	0.900*	58.2	543.8	0.869*	638.3	9.024	0.890*	10.905	53.336	0.923*	59.592
stack	53.5	0.903*	58.6	539.2	0.850*	627.9	9.318	0.895*	10.090	54.761	0.892*	67.272
Never helpful at any size (ratio only):												
arc fixing		1.200†			1.168†			1.113†			1.192†	
arc fixing + renumb.		1.290†			1.396†			1.506†			1.475†	
cost GCD		0.985			1.009			1.001			0.955	
node presolve		1.017			1.000			1.006			0.961	
arc presolve		0.990			0.999			1.013			0.985	
no arc reversal		0.984			0.996			0.997			1.052	
path length $\ell = 2$		1.114†			1.052†			1.047†			1.145†	
highest excess		1.208†			1.217†			1.097†			1.064	
renumbering		1.171†			1.236†			1.169†			1.226†	

Table 23: Heuristics for benchmark family `netgen_sr` under NS; the sizes are those of Table 2.

	$k = 10$ [ms] $n = 1024$ $m = 32\,768$			$k = 12$ [ms] $n = 4096$ $m = 262\,144$			$k = 14$ [ms] $n = 16\,384$ $m = 2\,097\,152$			$k = 16$ [s] $n = 65\,536$ $m = 16\,777\,216$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
no arc reversal	4.9	0.974*	5.1	55.2	1.070	53.5	787.8	1.017	751.4	11.831	1.076	11.570
no arc mixing	4.8	0.936*	5.1	49.9	0.946*	53.5	704.8	0.930*	751.4	10.876	0.957*	11.570
stack	4.4	0.896*	5.0		–			–			–	
Never helpful at any size (ratio only):												
renumbering		1.039†			1.092†			1.132†			1.915†	
node presolve		0.991			0.999			0.994			1.070†	
arc presolve		0.994			1.000			1.001			1.024†	
anti-stalling		4.114†			6.911†			11.773†			DNF	
no node reversal		0.998			0.999			1.000			0.979	
anti-stalling		–			–			–			17.095†	

Table 24: Heuristics for benchmark family `netgen_sr` under COS; the sizes are those of Table 2.

	$k = 10$ [ms] $n = 1024$ $m = 32\,768$			$k = 12$ [ms] $n = 4096$ $m = 262\,144$			$k = 14$ [s] $n = 16\,384$ $m = 2\,097\,152$			$k = 16$ [s] $n = 65\,536$ $m = 16\,777\,216$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
no arc reversal	12.2	0.995	12.2	94.1	1.013	92.6	1.577	1.050	1.501	12.924	0.972*	13.397
path length $\ell = 2$	12.2	1.023	12.2	85.0	0.935*	92.3	1.481	1.012	1.524	14.650	1.053	13.795
Never helpful at any size (ratio only):												
arc fixing		1.227†			1.461†			1.658†			1.975†	
arc fixing + renumb.		1.549†			1.949†			2.099†			DNF	
discharge		0.989			1.031†			1.038			1.000	
$\alpha = 20$		0.924			0.997			1.009			1.003	
$\alpha = 12$		0.986			1.030†			0.996			0.961	
cost GCD		0.983			1.001			0.979			1.004	
node presolve		0.994			0.990			1.011			0.992	
arc presolve		0.987			1.005			1.039			0.994	
reverse nodes		0.945			1.013			1.062			1.001	
reverse core arcs		1.006			1.038			1.084†			0.994	
arc fixing + renumb.		–			–			–			2.767†	
path length $\ell = 8$		0.991			1.003			1.009			1.015	
highest excess		1.109†			1.157†			1.167†			1.137†	
renumbering		1.289†			1.610†			1.470†			1.648†	

Table 25: Heuristics for benchmark family `goto_8` under NS; the sizes are those of Table 4.

	$k = 13$ [ms] $n = 8192$ $m = 65\,536$			$k = 19$ [s] $n = 524\,288$ $m = 4\,194\,304$			$k = 21$ [ms] $n = 2\,097\,152$ $m = 16\,777\,216$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
no arc mixing	310.4	0.999	309.8	2205.992	$\boxed{0.947}$*	2338.593	DNF		
Never helpful at any size (ratio only):									
renumbering		1.015			1.016			DNF	
node presolve		1.014			1.021†			DNF	
arc presolve		1.020†			0.993			DNF	
anti-stalling		1.007			1.014			DNF	
no node reversal		0.992			1.014			DNF	
no arc reversal		0.998			1.029†			DNF	

Table 26: Heuristics for benchmark family `goto_8` under COS; the sizes are those of Table 4. At $k = 13$, $k = 16$ and $k = 19$ the phase marks that delimit Init+loop did not fire, so the affected runs are scored on the entire compute time instead.

	$k = 13$ [ms] $n = 8192$ $m = 65\,536$			$k = 16$ [s] $n = 65\,536$ $m = 524\,288$			$k = 19$ [s] $n = 524\,288$ $m = 4\,194\,304$			$k = 21$ [s] $n = 2\,097\,152$ $m = 16\,777\,216$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
discharge	284.6	1.000	284.6	5.229	0.997	5.322	362.635	0.876*	414.385	851.367	0.968	822.708
$\alpha = 20$	287.0	1.037	284.6	5.293	0.985	5.322	308.083	$\boxed{0.743}$*	414.385	786.039	0.908*	822.708
reverse nodes	284.8	1.009	284.6	5.426	1.020	5.322	424.154	0.934*	414.385	798.104	1.015	822.708
reverse core arcs	264.8	$\boxed{0.930}$*	284.6	5.233	0.983	5.322	421.000	0.970	414.385	889.566	1.116	822.708
path length $\ell = 8$	304.0	1.097	279.5	5.701	1.004	5.366	388.730	0.930*	422.067	748.474	0.920*	813.137
stack		–			–		291.743	0.779*	379.935	705.467	$\boxed{0.850}$*	831.300
Never helpful at any size (ratio only):												
arc fixing		1.031†			1.093†			1.178†			1.011	
arc fixing + renumb.		1.099†			1.280†			1.465†			0.949	
$\alpha = 12$		1.002			1.163†			1.112†			DNF	
cost GCD		0.985			1.005			0.980			1.031	
node presolve		1.956†			2.025†			1.913†			0.957	
arc presolve		1.006			1.013			0.993			1.249†	
no arc reversal		1.177†			1.258†			1.666†			1.148†	
path length $\ell = 2$		1.054†			1.097†			1.029†			1.320†	
highest excess		2.812†			3.845†			CAP			CAP	
renumbering		1.097†			1.191†			1.310†			0.983	

Table 27: Heuristics for benchmark family `goto_8` under CAS; the sizes are those of Table 4.

	$k = 16$ [s] $n = 65\,536$ $m = 524\,288$		
Heuristic	Time	Ratio	Calib.
radix heap	20.132	0.740*	28.602
hot queue	25.387	0.866*	28.602
multi-source Dijkstra	27.905	0.965*	28.602
renumbering	20.750	0.772*	28.602
no node reversal	24.585	0.828*	28.602
stack	16.508	$\boxed{0.576}$*	28.657
Never helpful at any size (ratio only):			
label reuse		1.150	
node presolve		0.998	
arc presolve		0.962	
no arc reversal		0.982	

Table 28: Heuristics for benchmark family `goto_sr` under NS; the sizes are those of Table 5.

	$k = 10$ [ms] $n = 1024$ $m = 32\,768$			$k = 12$ [ms] $n = 4096$ $m = 262\,144$			$k = 14$ [s] $n = 16\,384$ $m = 2\,097\,152$			$k = 16$ [s] $n = 65\,536$ $m = 16\,777\,216$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
arc presolve	29.5	0.997	29.5	757.7	1.011	746.1	20.139	0.987	20.462	750.857	0.824*	888.173
no arc mixing	30.8	1.047	29.5	753.7	1.010	746.1	19.794	0.968*	20.462	657.247	0.763*	888.173
stack		–			–			–		705.479	0.802*	877.634
Never helpful at any size (ratio only):												
renumbering		1.027†			1.063†			1.157†			1.112†	
node presolve		0.999			1.008			1.017			1.005	
anti-stalling		1.007			1.006			1.004			1.041†	
no node reversal		0.999			0.997			1.004			0.997	
no arc reversal		1.006			1.000			1.003			1.032	

Table 29: Heuristics for benchmark family `goto_sr` under COS; the sizes are those of Table 5. At $k = 10$, $k = 14$ and $k = 16$ the phase marks that delimit Init+loop did not fire, so the affected runs are scored on the entire compute time instead.

	$k = 10$ [ms] $n = 1024$ $m = 32\,768$			$k = 12$ [ms] $n = 4096$ $m = 262\,144$			$k = 14$ [s] $n = 16\,384$ $m = 2\,097\,152$			$k = 16$ [s] $n = 65\,536$ $m = 16\,777\,216$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
arc fixing	72.6	1.052	69.5	696.4	1.012	691.1	14.784	1.035	15.012	195.977	0.954*	199.692
$\alpha = 12$	67.2	0.967*	69.5	635.1	0.919*	691.1	13.878	0.959*	15.012	187.560	0.939*	199.692
reverse core arcs	70.6	1.016	69.5	676.1	0.977*	691.1	13.370	0.947*	15.012	192.673	0.991	199.692
path length $\ell = 2$	68.5	1.001	69.1	654.2	0.968	690.3	15.168	0.917*	15.955	209.334	0.909*	226.863
stack		–		631.8	0.894*	696.5	14.123	0.942*	15.052	227.534	0.905*	251.355
Never helpful at any size (ratio only):												
arc fixing + renumb.		1.111†			1.069†			1.018			1.062†	
discharge		1.007			1.021†			1.023			0.972	
$\alpha = 20$		1.066†			1.086†			1.067			1.077	
cost GCD		0.994			0.991			1.009			0.981	
node presolve		1.928†			0.996			1.986†			2.019†	
arc presolve		1.058†			1.008			1.042†			1.034	
no arc reversal		1.116†			1.115†			1.043			1.072†	
reverse nodes		1.036†			1.016			1.009			1.004	
path length $\ell = 8$		1.217†			1.209†			1.212†			1.343†	
highest excess		1.595†			1.955†			2.105†			CAP	
renumbering		1.074†			1.073†			1.022†			1.083†	

Table 30: Heuristics for benchmark family `grid_wide` under NS; the sizes are those of Table 6.

	$k = 13$ [ms] $n = 8194$ $m = 16\,880$			$k = 16$ [ms] $n = 65\,538$ $m = 135\,152$			$k = 18$ [ms] $n = 262\,146$ $m = 540\,656$			$k = 19$ [ms] $n = 524\,290$ $m = 1\,081\,328$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
Never helpful at any size (ratio only):												
renumbering		1.238†			2.195†			5.060†			DNF	
node presolve		1.001			0.991			0.997			0.993	
arc presolve		1.000			0.994			0.999			0.999	
anti-stalling		18.067†			94.079†			332.885†			DNF	
no node reversal		0.999			0.991			0.997			1.019	
no arc reversal		1.002			1.007			1.001			1.015	
no arc mixing		1.045†			1.130†			1.148†			1.109†	
renumbering		–			–			–			6.370†	
anti-stalling		–			–			–			CAP	

Table 31: Heuristics for benchmark family `grid_wide` under COS; the sizes are those of Table 6.

	$k = 13$ [ms] $n = 8194$ $m = 16\,880$			$k = 16$ [s] $n = 65\,538$ $m = 135\,152$			$k = 18$ [s] $n = 262\,146$ $m = 540\,656$			$k = 19$ [s] $n = 524\,290$ $m = 1\,081\,328$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
arc fixing + renumb.	78.4	1.124	67.3	1.159	1.102	1.068	13.432	0.974	13.851	49.184	0.915*	53.307
$\alpha = 12$	66.7	0.975	67.3	1.027	0.967*	1.068	13.200	0.951*	13.851	55.834	0.941	53.307
cost GCD	66.6	0.990	67.3	0.997	0.950*	1.068	13.213	0.954*	13.851	51.727	0.964*	53.307
node presolve	67.3	0.985	67.3	1.216	0.936*	1.068	12.319	0.939*	13.851	53.236	0.972*	53.307
arc presolve	66.5	0.987	67.3	0.998	0.945*	1.068	13.072	0.921*	13.851	53.508	0.937*	53.307
no arc reversal	64.7	0.954*	67.3	0.979	0.911*	1.068	12.443	0.889*	13.851	48.154	0.850*	53.307
path length $\ell = 8$	64.8	0.936*	66.9	1.462	1.173	1.052	12.639	1.092	12.532	70.803	1.292	54.897
renumbering	72.4	1.025	66.9	1.107	1.016	1.052	14.162	0.984	12.532	49.375	0.888*	54.897
stack	60.3	0.887*	66.7	0.902	0.806*	1.061	9.901	0.740*	13.467	42.662	0.785*	54.013
Never helpful at any size (ratio only):												
arc fixing		1.059†			1.074†			1.056†			1.017	
discharge		1.000			1.023†			1.015			1.046	
$\alpha = 20$		1.022†			1.004			1.110†			1.107†	
reverse nodes		0.994			1.012			1.036			0.990	
reverse core arcs		1.022			0.999			0.953			0.972	
path length $\ell = 2$		1.112†			1.144†			1.104†			1.132†	
highest excess		1.436†			1.381			1.322†			1.195†	

Table 32: Heuristics for benchmark family `grid_square` under NS; the sizes are those of Table 7.

	$k = 13$ [ms] $n = 8283$ $m = 16\,562$			$k = 16$ [s] $n = 65\,538$ $m = 131\,072$			$k = 18$ [s] $n = 262\,146$ $m = 524\,288$			$k = 19$ [s] $n = 524\,178$ $m = 1\,048\,352$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
arc presolve	27.5	1.018	27.0	1.397	0.993	1.407	28.829	0.992	29.041	124.449	0.965*	128.851
no arc reversal	28.2	1.052	27.0	1.415	1.017	1.407	26.981	0.932*	29.041	111.554	0.858*	128.851
stack		–			–			–		107.826	0.821*	129.009
Never helpful at any size (ratio only):												
renumbering		1.070†			1.096†			1.115†			1.135†	
node presolve		0.996			0.999			0.998			0.983	
anti-stalling		9.222†			18.546†			30.248†			DNF	
no node reversal		0.994			1.001			0.997			0.995	
no arc mixing		1.047†			1.100†			1.169†			1.141†	
anti-stalling		–			–			–			CAP	

Table 33: Heuristics for benchmark family `grid_square` under COS; the sizes are those of Table 7.

	$k = 13$ [ms] $n = 8283$ $m = 16\,562$			$k = 16$ [s] $n = 65\,538$ $m = 131\,072$			$k = 18$ [s] $n = 262\,146$ $m = 524\,288$			$k = 19$ [s] $n = 524\,178$ $m = 1\,048\,352$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
$\alpha = 12$	68.8	1.069	68.4	1.308	1.188	1.137	10.445	0.960*	10.845	30.752	0.991	30.346
node presolve	67.6	0.992	68.4	1.114	0.980*	1.137	10.766	0.994	10.845	30.728	1.013	30.346
no arc reversal	63.6	0.956*	68.4	1.079	0.968	1.137	10.886	1.006	10.845	30.569	1.007	30.346
reverse nodes	65.7	0.969*	68.4	1.135	1.024	1.137	10.978	1.006	10.845	29.928	0.992	30.346
reverse core arcs	71.7	1.010	68.4	1.186	1.025	1.137	10.852	0.977*	10.845	31.240	1.009	30.346
path length $\ell = 8$	65.3	0.945*	69.0	1.072	0.993	1.123	9.647	0.909*	10.492	31.392	0.991	30.013
stack	62.2	0.936*	68.5		–		9.575	0.918*	10.402		–	
Never helpful at any size (ratio only):												
arc fixing		1.078†			1.130†			1.050†			1.039†	
arc fixing + renumb.		1.124†			1.180†			1.190†			1.186†	
discharge		0.999			1.001			1.004			1.049†	
$\alpha = 20$		0.980			0.991			1.005			1.026	
cost GCD		0.992			0.995			1.007			1.003	
arc presolve		0.995			0.986			1.028†			0.999	
path length $\ell = 2$		1.114†			1.086			1.140†			1.188†	
highest excess		1.623†			2.431†			2.641†			2.890†	
renumbering		1.054†			1.115†			1.093†			1.095†	

Table 34: Heuristics for benchmark family `grid_square` under CAS; the sizes are those of Table 7.

	$k = 19$ [s] $n = 524\,178$ $m = 1\,048\,352$		
Heuristic	Time	Ratio	Calib.
radix heap	48.566	0.652*	88.339
label reuse	81.597	0.924*	88.339
Never helpful at any size (ratio only):			
hot queue		1.159†	
multi-source Dijkstra		1.003	
renumbering		1.456†	
node presolve		1.003	
arc presolve		1.006	
no arc reversal		1.049	
no node reversal		2.207†	

Table 35: Heuristics for benchmark family `grid_long` under CAS; the sizes are those of Table 8.

	$k = 13$ [ms] $n = 8194$ $m = 15\,888$			$k = 16$ [ms] $n = 65\,538$ $m = 126\,992$			$k = 18$ [ms] $n = 262\,146$ $m = 507\,920$			$k = 19$ [ms] $n = 524\,290$ $m = 1\,015\,824$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
radix heap	10.2	0.647*	14.5	55.0	0.697*	78.8	172.3	0.613*	272.9	337.6	0.514*	650.8
hot queue	15.7	0.991	14.5	83.2	1.040	78.8	258.5	0.942*	272.9	517.3	0.795*	650.8
multi-source Dijkstra	13.8	0.960*	14.5	78.3	0.993	78.8	271.9	0.997	272.9	653.7	0.975*	650.8
label reuse	15.3	0.965*	14.5	76.6	0.981	78.8	267.7	1.006	272.9	649.8	0.958*	650.8
node presolve	14.6	0.999	14.5	77.1	0.988	78.8	274.6	0.999	272.9	650.4	0.977*	650.8
factor 8	15.3	1.081	14.5	69.5	0.885	79.2	248.3	0.893*	265.2	705.5	1.017	707.7
stack	9.7	0.669*	14.4		–		144.3	0.501*	263.3	355.7	0.531*	659.7
Never helpful at any size (ratio only):												
renumbering		1.037†			1.201†			1.334†			1.434†	
arc presolve		1.002			0.995			0.990			0.988	
no arc reversal		1.021†			1.022†			1.023†			1.004	
no node reversal		1.020			0.994			0.995			0.983	
blocking flow		4.200†			4.238†			1.179†			1.136†	
factor 2		0.901			0.860			0.929			1.017	

Table 36: Heuristics for benchmark family `grid_long` under SSP; the sizes are those of Table 8.

	$k = 13$ [ms] $n = 8194$ $m = 15\,888$			$k = 16$ [ms] $n = 65\,538$ $m = 126\,992$			$k = 18$ [ms] $n = 262\,146$ $m = 507\,920$			$k = 19$ [ms] $n = 524\,290$ $m = 1\,015\,824$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
radix heap	19.6	0.619*	30.6	67.3	0.631*	111.0	168.1	0.633*	262.8	346.2	0.516*	659.8
hot queue	32.7	0.973	30.6	102.0	0.968*	111.0	264.0	0.948*	262.8	529.8	0.780*	659.8
label reuse	30.9	0.964*	30.6	109.0	0.992	111.0	260.3	0.997	262.8	656.4	0.971	659.8
stack	19.1	0.596*	30.5		–			–			–	
Never helpful at any size (ratio only):												
multi-source Dijkstra		1.002			0.996			1.000			1.016	
node presolve		1.001			0.991			0.998			0.986	
arc presolve		0.999			1.001			0.997			1.028	
no arc reversal		1.012			1.019			1.027†			1.027	
no node reversal		1.000			0.996			1.000			1.008	
blocking flow		1.497†			1.295†			1.169†			1.234†	

Table 37: Heuristics for benchmark family `road_paths` under CAS; the sizes are those of Table 9.

	`NH` [ms] $n = 116\,920$ $m = 265\,402$			`NV` [ms] $n = 261\,155$ $m = 620\,924$			`WI` [s] $n = 519\,157$ $m = 1\,266\,876$			`TX` [s] $n = 2\,073\,870$ $m = 5\,156\,088$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
radix heap	51.4	0.888*	57.9	201.1	0.834*	231.1	1.120	0.853*	1.328	5.058	0.745*	6.903
multi-source Dijkstra	55.8	0.977*	57.9	228.8	1.007	231.1	1.229	0.962*	1.328	6.452	0.963*	6.903
node presolve	55.8	0.969*	57.9	224.8	0.973*	231.1	1.325	1.016	1.328	7.072	1.016	6.903
arc presolve	58.7	0.998	57.9	228.3	0.980*	231.1	1.326	1.007	1.328	7.253	1.050	6.903
no node reversal	69.3	1.097	57.9	238.2	1.068	231.1	1.134	0.920*	1.328	6.982	1.031	6.903
stack	49.3	0.832*	59.6	193.1	0.807*	239.2	0.836	0.761*	1.238	5.623	0.726*	7.904
Never helpful at any size (ratio only):												
hot queue		1.138†			1.288†			1.334†			1.216†	
label reuse		1.004			1.004			1.000			0.998	
renumbering		0.999			1.006			1.002			1.055	
no arc reversal		1.039†			1.007			1.018			1.043†	
blocking flow		5.573†			5.642†			6.565†			9.732†	
factor 2		1.002			0.995			1.002			1.001	
factor 8		1.000			0.993			0.988			0.977	

Table 38: Heuristics for benchmark family `road_paths` under SSP; the sizes are those of Table 9.

	`NH` [ms] $n = 116\,920$ $m = 265\,402$			`NV` [ms] $n = 261\,155$ $m = 620\,924$			`WI` [s] $n = 519\,157$ $m = 1\,266\,876$			`TX` [s] $n = 2\,073\,870$ $m = 5\,156\,088$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
radix heap	52.2	0.891*	58.6	195.2	0.833*	234.3	1.119	0.854*	1.294	5.544	0.812*	7.080
multi-source Dijkstra	56.3	0.965*	58.6	245.2	1.039	234.3	1.227	0.971*	1.294	6.851	0.988	7.080
node presolve	56.3	0.955*	58.6	233.6	0.982	234.3	1.353	1.019	1.294	7.370	1.023	7.080
arc presolve	56.4	0.963*	58.6	229.6	0.977*	234.3	1.329	1.027	1.294	7.145	1.009	7.080
no node reversal	69.0	1.088	58.6	240.1	1.068	234.3	1.128	0.926*	1.294	7.422	0.989	7.080
stack	49.0	0.821*	59.1	198.2	0.807*	236.2	0.814	0.754*	1.134		–	
Never helpful at any size (ratio only):												
hot queue		1.109†			1.279†			1.310†			1.277†	
label reuse		0.989			0.997			0.998			1.021	
no arc reversal		1.015			1.014			1.021†			0.981	
blocking flow		5.682†			5.555†			6.596†			9.009†	

Table 39: Heuristics for benchmark family `road_flow` under CAS; the sizes are those of Table 10.

	`NH` [ms] $n = 116\,920$ $m = 265\,402$			`NV` [ms] $n = 261\,155$ $m = 620\,924$			`WI` [s] $n = 519\,157$ $m = 1\,266\,876$			`TX` [s] $n = 2\,073\,870$ $m = 5\,156\,088$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
radix heap	242.8	0.809*	301.6	629.8	0.850*	740.7	1.898	0.885*	2.191	18.386	0.825*	20.919
multi-source Dijkstra	280.7	0.947	301.6	752.3	0.962	740.7	2.234	0.971*	2.191	20.222	1.011	20.919
factor 2	267.0	0.893	299.9	638.7	0.861*	747.4	2.556	1.231	2.128	26.058	1.265	21.401
stack		–		506.9	0.704*	762.2	1.782	0.824*	2.164		–	
Never helpful at any size (ratio only):												
hot queue		1.463†			1.660†			1.618†			DNF	
label reuse		0.996			0.997			0.996			1.007	
renumbering		1.262†			1.447†			1.399†			DNF	
node presolve		0.980			0.994			1.013			1.020	
arc presolve		0.982			1.000			0.999			0.995	
no arc reversal		1.031†			1.027†			1.013			1.019	
no node reversal		0.976			1.107†			1.047			1.188	
hot queue		–			–			–			1.894†	
renumbering		–			–			–			1.499†	
blocking flow		58.552†			59.434†			CAP			CAP	
factor 8		0.995			0.995			1.006			1.001	

Table 40: Heuristics for benchmark family `road_flow` under SSP; the sizes are those of Table 10.

	`NH` [ms] $n = 116\,920$ $m = 265\,402$			`NV` [ms] $n = 261\,155$ $m = 620\,924$			`WI` [s] $n = 519\,157$ $m = 1\,266\,876$			`TX` [s] $n = 2\,073\,870$ $m = 5\,156\,088$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
radix heap	171.7	0.801*	214.4	512.2	0.804*	630.1	2.950	0.801*	3.526	19.046	0.758*	25.870
multi-source Dijkstra	212.1	0.987	214.4	609.7	0.996	630.1	3.699	1.003	3.526	24.202	0.946*	25.870
stack		–			–			–		20.470	0.720*	28.449
Never helpful at any size (ratio only):												
hot queue		1.326†			1.415†			1.338†			1.195†	
label reuse		0.995			1.007			1.012			0.990	
node presolve		0.985			0.982			1.018			1.000	
arc presolve		0.980			0.986			1.002			0.995	
no arc reversal		1.014			1.021†			1.021†			1.031†	
no node reversal		1.061			1.128†			0.990			1.113†	
blocking flow		4.150†			5.034†			7.112†			6.917†	

Table 41: Heuristics for benchmark family `vision_rnd` under NS; the sizes are those of Table 11.

	`bone_subxyz_subx` [s] $n = 491\,522$ $m = 2\,877\,382$			`bone_subxyz` [s] $n = 983\,042$ $m = 5\,776\,516$			`bone_subxy` [s] $n = 1\,949\,698$ $m = 11\,520\,153$			`bone_subx` [s] $n = 3\,899\,394$ $m = 23\,091\,149$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
node presolve	13.700	0.944*	14.498	107.423	1.012	102.276	686.535	1.004	684.818	3955.816	0.988	4010.470
no arc reversal	14.396	0.989	14.498	103.332	0.987	102.276	665.119	0.981	684.818	3779.049	0.919*	4010.470
no arc mixing	9.975	0.688*	14.498	63.152	0.582*	102.276	212.398	0.310*	684.818	762.235	0.188*	4010.470
stack	10.007	0.668*	14.468		–			–		764.012	0.174*	4242.629
Never helpful at any size (ratio only):												
renumbering		1.870†			1.727†			1.326†			1.178†	
arc presolve		0.981			1.068†			1.070			1.019	
anti-stalling		77.296†			DNF			DNF			DNF	
no node reversal		0.991			1.011			0.952			0.987	

Table 42: Heuristics for benchmark family `vision_rnd` under COS; the sizes are those of Table 11.

	`bone_subxyz_subx` [s] $n = 491\,522$ $m = 2\,877\,382$			`bone_subxyz` [s] $n = 983\,042$ $m = 5\,776\,516$			`bone_subxy` [s] $n = 1\,949\,698$ $m = 11\,520\,153$			`bone_subx` [s] $n = 3\,899\,394$ $m = 23\,091\,149$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
discharge	13.947	1.009	13.903	51.106	0.960*	58.660	117.496	0.966	116.900	309.990	1.000	310.042
path length $\ell = 8$	13.939	0.981	14.329	59.246	0.971*	58.263	131.766	0.953*	141.582	323.776	0.930	329.258
stack		–		58.289	0.964*	60.325		–			–	
Never helpful at any size (ratio only):												
arc fixing		1.144†			1.060†			1.028†			1.165†	
arc fixing + renumb.		1.166†			1.104†			1.172†			1.200†	
$\alpha = 20$		0.999			1.003			0.973			1.055	
$\alpha = 12$		1.161†			1.059†			1.028†			1.004	
cost GCD		1.001			1.000			0.995			1.020	
node presolve		1.001			0.997			0.961			0.952	
arc presolve		1.006			1.007			0.976			0.932	
no arc reversal		1.014			1.066†			1.029			0.996	
reverse nodes		1.041†			0.988			1.010			0.985	
reverse core arcs		1.140†			1.012			0.980			0.903	
path length $\ell = 2$		1.441†			1.175†			1.185†			1.275†	
highest excess		1.671†			1.488†			1.414†			1.380†	
renumbering		1.117†			1.105†			1.120†			1.078†	

Table 43: Heuristics for benchmark family `vision_prop` under NS; the sizes are those of Table 12.

	`bone_subxyz_subx` [s] $n = 491\,522$ $m = 2\,877\,382$			`bone_subxyz` [s] $n = 983\,042$ $m = 5\,776\,516$			`bone_subxy` [s] $n = 1\,949\,698$ $m = 11\,520\,153$			`bone_subx` [s] $n = 3\,899\,394$ $m = 23\,091\,149$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
node presolve	14.376	0.975*	14.949	83.455	1.005	81.956	411.926	0.989	413.130	1922.267	0.945	2078.986
no arc reversal	13.816	0.937*	14.949	78.327	0.976*	81.956	407.482	0.986	413.130	1998.093	0.992	2078.986
no arc mixing	12.718	0.859*	14.949	65.962	0.792*	81.956	170.917	0.393*	413.130	535.636	0.259*	2078.986
stack	12.172	0.816*	14.845	55.182	0.696*	83.454		–			–	
Never helpful at any size (ratio only):												
renumbering		1.691†			1.804†			1.740†			1.444†	
arc presolve		0.997			1.058†			1.066†			1.033	
anti-stalling		60.200†			DNF			DNF			DNF	
no node reversal		1.008			0.992			0.999			1.010	

Table 44: Heuristics for benchmark family `vision_prop` under COS; the sizes are those of Table 12.

	`bone_subxyz_subx` [s] $n = 491\,522$ $m = 2\,877\,382$			`bone_subxyz` [s] $n = 983\,042$ $m = 5\,776\,516$			`bone_subxy` [s] $n = 1\,949\,698$ $m = 11\,520\,153$			`bone_subx` [s] $n = 3\,899\,394$ $m = 23\,091\,149$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
$\alpha = 12$	29.482	0.958*	30.525	102.605	0.937	103.090	252.310	0.939*	268.683	575.585	0.969	594.070
Never helpful at any size (ratio only):												
arc fixing		1.074†			1.059†			1.022			1.020	
arc fixing + renumb.		1.200†			1.123†			1.184†			1.237†	
discharge		1.018			1.051†			0.973			0.995	
$\alpha = 20$		1.126†			1.126†			1.045†			1.054†	
cost GCD		1.000			1.060			0.991			1.000	
node presolve		1.015			1.066†			0.976			1.020	
arc presolve		1.005			1.071†			0.996			1.000	
no arc reversal		1.054†			1.017			1.036†			1.038†	
reverse nodes		1.008			1.025			1.014			1.007	
reverse core arcs		1.053			1.004			0.954			1.015	
path length $\ell = 2$		1.124†			1.062†			1.171†			1.213†	
path length $\ell = 8$		1.087			1.014			1.042			1.036†	
highest excess		1.521†			1.413†			1.493†			1.379†	
renumbering		1.130†			1.033†			1.097†			1.191†	

B.2 Additional Benchmarks

Table 45: Heuristics for benchmark family `gridgen_8` under NS; the sizes are those of Table 13.

	$k = 13$ [ms] $n = 8191$ $m = 65\,528$			$k = 16$ [ms] $n = 65\,537$ $m = 524\,296$			$k = 19$ [s] $n = 524\,177$ $m = 4\,193\,416$			$k = 21$ [s] $n = 2\,096\,705$ $m = 16\,773\,640$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
no arc reversal	39.5	0.957	39.9	778.3	0.977*	802.0	23.655	0.990	22.609	191.906	0.958	192.388
no arc mixing	39.2	0.976*	39.9	822.1	1.036	802.0	23.414	1.042	22.609	185.001	0.916*	192.388
Never helpful at any size (ratio only):												
renumbering		0.988			1.022†			1.791†			DNF	
node presolve		0.989			0.982			1.006			1.018	
arc presolve		0.995			1.006			1.007			0.954	
anti-stalling		32.336†			136.172†			DNF			DNF	
no node reversal		0.998			0.988			1.012			1.052	

Table 46: Heuristics for benchmark family `gridgen_8` under COS; the sizes are those of Table 13.

	$k = 13$ [ms] $n = 8191$ $m = 65\,528$			$k = 16$ [ms] $n = 65\,537$ $m = 524\,296$			$k = 19$ [s] $n = 524\,177$ $m = 4\,193\,416$			$k = 21$ [s] $n = 2\,096\,705$ $m = 16\,773\,640$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
$\alpha = 20$	59.5	1.016	58.5	585.6	1.042	554.6	8.992	1.140	7.886	47.763	0.952*	51.581
$\alpha = 12$	57.2	0.977	58.5	567.5	0.985	554.6	7.771	0.983	7.886	47.836	0.932*	51.581
no arc reversal	57.1	0.976*	58.5	553.6	0.978*	554.6	7.206	0.911*	7.886	46.304	0.898*	51.581
path length $\ell = 8$	58.3	1.003	58.3	522.3	0.976*	551.9	8.746	0.991	8.962	40.825	0.874*	47.855
stack		–		531.3	0.933*	569.2		–		40.765	0.818*	49.834
Never helpful at any size (ratio only):												
arc fixing		1.366†			1.135†			1.255†			1.189†	
arc fixing + renumb.		1.502†			1.401†			1.663†			1.705†	
discharge		0.995			1.012			0.993			1.033	
cost GCD		0.986			1.015			0.989			1.006	
node presolve		0.993			1.003			0.993			0.998	
arc presolve		0.996			0.984			0.985			0.997	
reverse nodes		1.038†			0.981			1.052†			0.942	
reverse core arcs		1.014			1.023†			1.054			1.008	
path length $\ell = 2$		1.114†			1.055			1.198†			1.061†	
highest excess		1.301†			1.274†			1.222†			0.960	
renumbering		1.312†			1.315†			1.395†			1.379†	

Table 47: Heuristics for benchmark family `gridgen_sr` under NS; the sizes are those of Table 14.

	$k = 10$ [ms] $n = 1025$ $m = 32\,800$			$k = 12$ [ms] $n = 4097$ $m = 262\,208$			$k = 14$ [ms] $n = 16\,385$ $m = 2\,097\,280$			$k = 16$ [s] $n = 65\,537$ $m = 16\,777\,472$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
no arc mixing	5.2	0.985	5.3	48.2	0.935*	48.7	572.2	1.020	597.9	8.297	1.035	8.017
Never helpful at any size (ratio only):												
renumbering		1.047†			1.112†			1.220†			2.070†	
node presolve		1.004			0.998			1.005			0.992	
arc presolve		0.992			1.004			1.009			0.913	
anti-stalling		4.486†			7.347†			12.558†			DNF	
no node reversal		1.002			1.002			1.010			1.020	
no arc reversal		1.104			1.028			1.026			1.025	
anti-stalling		–			–			–			25.509†	

Table 48: Heuristics for benchmark family `gridgen_sr` under COS; the sizes are those of Table 14.

	$k = 10$ [ms] $n = 1025$ $m = 32\,800$			$k = 12$ [ms] $n = 4097$ $m = 262\,208$			$k = 14$ [s] $n = 16\,385$ $m = 2\,097\,280$			$k = 16$ [s] $n = 65\,537$ $m = 16\,777\,472$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
discharge	12.8	0.980*	13.1	99.2	0.979	99.5	1.477	0.961*	1.537	15.451	1.003	15.060
$\alpha = 20$	13.1	0.994	13.1	96.2	0.954*	99.5	1.485	0.966	1.537	15.245	1.009	15.060
cost GCD	12.8	0.979*	13.1	98.3	0.985	99.5	1.496	0.972	1.537	15.339	1.016	15.060
node presolve	12.9	0.983	13.1	98.2	0.990	99.5	1.591	0.989	1.537	15.207	0.961*	15.060
arc presolve	12.8	0.975*	13.1	97.5	0.979*	99.5	1.564	0.972	1.537	15.356	1.008	15.060
reverse core arcs	13.3	0.940*	13.1	101.2	0.978	99.5	1.409	0.917*	1.537	15.216	1.038	15.060
path length $\ell = 2$	13.0	0.955	13.0	101.0	0.988	100.0	1.595	0.952*	1.765	15.031	0.962	15.287
stack	13.2	0.932*	12.9	95.7	0.953*	100.0	1.452	0.953*	1.524		–	
Never helpful at any size (ratio only):												
arc fixing		1.378†			2.113†			1.375†			1.802†	
arc fixing + renumb.		1.550†			2.220†			1.985†			DNF	
$\alpha = 12$		0.955			1.034†			0.945			0.992	
no arc reversal		0.986			0.986			0.980			1.015	
reverse nodes		0.977			0.998			0.979			0.997	
arc fixing + renumb.		–			–			–			3.219†	
path length $\ell = 8$		0.982			1.043†			1.012			1.011	
highest excess		1.305†			1.161†			1.220†			1.108†	
renumbering		1.295†			1.629†			1.499†			1.675†	

Table 49: Heuristics for benchmark family `gridgen_deg` under NS; the sizes are those of Table 15.

	$k = 5$ [ms] $n = 4097$ $m = 131\,104$			$k = 7$ [ms] $n = 4097$ $m = 524\,416$			$k = 9$ [ms] $n = 4097$ $m = 2\,097\,664$			$k = 11$ [ms] $n = 4097$ $m = 8\,390\,656$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
no arc reversal	30.8	1.049	29.4	77.3	0.953	76.7	196.9	1.027	193.1	593.4	0.957*	604.4
no arc mixing	32.6	1.048	29.4	74.2	0.949*	76.7	192.6	0.984	193.1	597.8	0.973*	604.4
stack		–			–			–		613.0	0.976*	606.0
Never helpful at any size (ratio only):												
renumbering		1.065†			1.161†			1.415†			1.950†	
node presolve		1.004			1.004			1.002			0.988	
arc presolve		0.998			1.001			1.012			DNF	
anti-stalling		10.832†			5.577†			8.268†			DNF	
no node reversal		1.006			1.001			1.007			0.966	
anti-stalling		–			–			–			11.296†	
arc presolve		–			–			–			0.995	

Table 50: Heuristics for benchmark family `gridgen_deg` under COS; the sizes are those of Table 15.

	$k = 5$ [ms] $n = 4097$ $m = 131\,104$			$k = 7$ [ms] $n = 4097$ $m = 524\,416$			$k = 9$ [s] $n = 4097$ $m = 2\,097\,664$			$k = 11$ [s] $n = 4097$ $m = 8\,390\,656$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
$\alpha = 20$	52.5	0.937*	56.8	188.1	1.024	192.5	1.491	1.078	1.399	6.139	1.060	5.802
$\alpha = 12$	60.3	1.002	56.8	199.9	1.013	192.5	1.535	1.029	1.399	5.623	0.955*	5.802
no arc reversal	56.3	0.991	56.8	190.1	0.996	192.5	1.402	0.992	1.399	5.718	0.963*	5.802
path length $\ell = 2$	58.4	0.956*	56.4	199.9	1.043	196.0	1.298	0.931	1.397	5.776	0.957	5.676
stack	58.3	0.992	56.8		–			–		5.712	0.949*	6.034
Never helpful at any size (ratio only):												
arc fixing		1.835†			2.265†			2.103†			2.331†	
arc fixing + renumb.		1.853†			2.514†			2.380†			DNF	
discharge		1.005			1.014			1.037			1.039	
cost GCD		0.993			0.989			0.997			1.011	
node presolve		0.991			0.994			0.991			0.976	
arc presolve		0.990			0.987			0.999			1.018	
reverse nodes		1.091†			1.017			1.020			0.977	
reverse core arcs		1.017			1.048†			0.967			1.040	
arc fixing + renumb.		–			–			–			2.902†	
path length $\ell = 8$		0.966			0.972			1.018			0.996	
highest excess		1.282†			1.195†			1.200†			1.048†	
renumbering		1.453†			1.812†			1.586†			1.783†	

Table 51: Heuristics for benchmark family `netgen_lo_8` under NS; the sizes are those of Table 16.

	$k = 13$ [ms] $n = 8192$ $m = 65\,536$			$k = 16$ [ms] $n = 65\,536$ $m = 524\,288$			$k = 19$ [s] $n = 524\,288$ $m = 4\,194\,304$			$k = 21$ [s] $n = 2\,097\,152$ $m = 16\,777\,216$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
node presolve	14.8	1.008	14.8	314.9	1.000	328.8	10.739	0.964*	11.166	102.565	1.016	107.032
no node reversal	14.7	1.002	14.8	325.3	0.993	328.8	10.915	0.978*	11.166	103.418	1.036	107.032
no arc reversal	15.6	1.071	14.8	322.8	1.023	328.8	10.840	0.978*	11.166	105.414	1.050	107.032
no arc mixing	15.0	1.073	14.8	330.4	1.039	328.8	10.847	0.971*	11.166	114.444	1.130	107.032
stack		–			–		10.208	0.979	10.542		–	
Never helpful at any size (ratio only):												
renumbering		1.111†			1.349†			2.612†			DNF	
arc presolve		1.002			0.994			1.008			0.979	
anti-stalling		10.023†			20.160†			25.642†			DNF	
renumbering		–			–			–			CAP	
anti-stalling		–			–			–			CAP	
stack		–			–			0.979			–	

Table 52: Heuristics for benchmark family `netgen_lo_8` under COS; the sizes are those of Table 16.

	$k = 13$ [ms] $n = 8192$ $m = 65\,536$			$k = 16$ [ms] $n = 65\,536$ $m = 524\,288$			$k = 19$ [s] $n = 524\,288$ $m = 4\,194\,304$			$k = 21$ [s] $n = 2\,097\,152$ $m = 16\,777\,216$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
$\alpha = 20$	26.0	0.903*	28.5	303.9	0.860*	354.3	6.502	0.940*	6.649	36.782	0.950	40.225
arc presolve	28.5	1.000	28.5	353.7	0.998	354.3	6.409	0.974*	6.649	39.670	0.991	40.225
no arc reversal	29.4	1.017	28.5	343.8	0.989	354.3	6.514	0.992	6.649	37.489	0.971*	40.225
reverse nodes	29.4	1.014	28.5	315.8	0.894*	354.3	6.618	1.021	6.649	40.026	1.002	40.225
reverse core arcs	27.7	0.967	28.5	315.0	0.891*	354.3	7.663	1.070	6.649	40.436	1.041	40.225
path length $\ell = 8$	25.9	0.901	28.4	284.1	0.833*	350.0	5.503	0.925*	5.948	35.644	0.885*	36.459
stack		–		290.7	0.842*	352.1	5.237	0.886*	6.090	39.832	0.912*	41.708
Never helpful at any size (ratio only):												
arc fixing		1.415†			1.410†			1.409†			1.424†	
arc fixing + renumb.		1.647†			1.859†			1.844†			1.893†	
discharge		0.984			1.002			0.988			0.990	
$\alpha = 12$		1.064†			1.062†			1.083†			1.035	
cost GCD		0.985			0.995			0.990			0.989	
node presolve		1.020†			1.015			0.990			0.980	
path length $\ell = 2$		1.093†			1.026†			1.081†			1.132	
highest excess		1.050†			0.996			1.079†			0.952	
renumbering		1.432†			1.507†			1.400†			1.442†	

Table 53: Heuristics for benchmark family `netgen_lo_sr` under NS; the sizes are those of Table 17.

	$k = 10$ [ms] $n = 1024$ $m = 32\,768$			$k = 12$ [ms] $n = 4096$ $m = 262\,144$			$k = 14$ [ms] $n = 16\,384$ $m = 2\,097\,152$			$k = 16$ [s] $n = 65\,536$ $m = 16\,777\,216$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
no arc reversal	2.1	1.048	2.0	19.5	0.993	19.3	230.4	1.027	217.2	2.800	0.958*	2.901
no arc mixing	1.8	0.895*	2.0	17.1	0.902*	19.3	197.3	0.888*	217.2	2.411	0.803*	2.901
stack		–			–			–		2.436	0.797*	3.034
Never helpful at any size (ratio only):												
renumbering		1.063†			1.167†			1.438†			3.006†	
node presolve		0.996			0.996			1.008			1.050†	
arc presolve		0.996			0.997			1.005			1.014	
anti-stalling		3.216†			3.742†			7.813†			DNF	
no node reversal		1.001			1.001			1.000			1.009	
anti-stalling		–			–			–			7.722†	

Table 54: Heuristics for benchmark family `netgen_lo_sr` under COS; the sizes are those of Table 17.

	$k = 10$ [ms] $n = 1024$ $m = 32\,768$			$k = 12$ [ms] $n = 4096$ $m = 262\,144$			$k = 14$ [s] $n = 16\,384$ $m = 2\,097\,152$			$k = 16$ [s] $n = 65\,536$ $m = 16\,777\,216$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
$\alpha = 20$	6.8	0.904*	7.1	56.5	0.986	58.4	0.948	0.949	1.005	9.530	0.909*	10.530
cost GCD	6.9	0.973*	7.1	58.5	1.006	58.4	1.057	1.026	1.005	10.755	1.004	10.530
arc presolve	6.9	0.975*	7.1	58.4	0.988	58.4	1.053	1.019	1.005	10.900	1.023	10.530
reverse core arcs	6.9	0.958*	7.1	58.8	1.014	58.4	0.984	0.986	1.005	9.885	0.946*	10.530
path length $\ell = 2$	7.0	0.960	7.1	57.7	0.981	58.8	0.993	0.936*	1.005	9.843	0.936*	10.478
stack	7.6	1.039	7.0		–			–		9.428	0.855*	11.151
Never helpful at any size (ratio only):												
arc fixing		1.377†			1.712†			1.799†			2.093†	
arc fixing + renumb.		1.871†			2.491†			2.470†			DNF	
discharge		0.992			0.999			1.052†			0.981	
$\alpha = 12$		1.001			1.078†			1.032†			1.026†	
node presolve		0.996			0.987			1.048†			1.004	
no arc reversal		1.005			1.034			1.021			0.986	
reverse nodes		1.045			1.021†			1.078			0.997	
arc fixing + renumb.		–			–			–			3.185†	
path length $\ell = 8$		1.072†			1.032†			1.008			1.011	
highest excess		1.043†			1.141†			1.156†			1.021†	
renumbering		1.547†			1.987†			1.599†			1.753†	

Table 55: Heuristics for benchmark family `vision_inv` under NS; the sizes are those of Table 18.

	`bone_subxyz_subx` [s] $n = 491\,522$ $m = 2\,877\,382$			`bone_subxyz` [s] $n = 983\,042$ $m = 5\,776\,516$			`bone_subxy` [s] $n = 1\,949\,698$ $m = 11\,520\,153$			`bone_subx` [s] $n = 3\,899\,394$ $m = 23\,091\,149$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
node presolve	11.300	0.993	11.486	53.422	0.988	54.653	363.140	0.964*	377.577	2107.825	0.931	2339.332
arc presolve	11.472	0.999	11.486	55.949	1.043	54.653	380.400	1.014	377.577	2261.956	0.937*	2339.332
no arc mixing	12.911	1.122	11.486	50.623	0.926*	54.653	98.147	0.260*	377.577	383.468	0.155*	2339.332
stack		–			–		95.474	0.250*	372.038	423.083	0.177*	2381.221
Never helpful at any size (ratio only):												
renumbering		1.756†			1.830†			1.519†			1.301†	
anti-stalling		116.537†			DNF			DNF			DNF	
no node reversal		1.015			0.990			0.983			1.014	
no arc reversal		1.051			1.045†			1.044†			1.184†	

Table 56: Heuristics for benchmark family `vision_inv` under COS; the sizes are those of Table 18.

	`bone_subxyz_subx` [s] $n = 491\,522$ $m = 2\,877\,382$			`bone_subxyz` [s] $n = 983\,042$ $m = 5\,776\,516$			`bone_subxy` [s] $n = 1\,949\,698$ $m = 11\,520\,153$			`bone_subx` [s] $n = 3\,899\,394$ $m = 23\,091\,149$		
Heuristic	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.	Time	Ratio	Calib.
discharge	28.244	1.006	28.260	82.308	0.948	87.963	187.049	0.947*	199.352	561.693	1.090	555.850
$\alpha = 20$	28.373	0.982	28.260	92.537	1.039	87.963	184.744	0.965*	199.352	562.837	1.032	555.850
$\alpha = 12$	27.766	1.010	28.260	85.275	0.911	87.963	161.653	0.886*	199.352	542.202	0.998	555.850
reverse core arcs	27.318	0.978*	28.260	89.897	1.022	87.963	188.031	0.989	199.352	479.502	1.006	555.850
stack		–			–		187.134	0.949*	224.125		–	
Never helpful at any size (ratio only):												
arc fixing		1.020			1.081			0.985			1.045†	
arc fixing + renumb.		1.145†			1.076			1.075†			1.270†	
cost GCD		1.014			1.027			0.989			1.014	
node presolve		1.025†			1.032			1.036†			1.018	
arc presolve		1.009			1.020			1.010			1.014	
no arc reversal		1.043†			1.129†			1.116†			1.130†	
reverse nodes		1.072			1.015			1.070†			0.994	
path length $\ell = 2$		1.221†			1.137†			1.317†			1.245†	
path length $\ell = 8$		1.172†			1.041†			1.064†			1.059†	
highest excess		1.664†			1.488†			1.549†			CAP	
renumbering		1.088†			1.107†			1.172†			1.119†	