Multistage Shortest Path: Instances and Practical Evaluation

Markus Chimani
Theoretical Computer Science, Universität Osnabrück, Germany

Niklas Troost
Theoretical Computer Science, Universität Osnabrück, Germany

Abstract

A multistage graph problem is a generalization of a traditional graph problem where, instead of a single input graph, we consider a sequence of graphs. We ask for a sequence of solutions, one for each input graph, such that consecutive solutions are as similar as possible. There are several theoretical results on different multistage problems and their complexities, as well as FPT and approximation algorithms. However, there is a severe lack of experimental validation and resulting feedback. Not only are there no algorithmic experiments in literature, we do not even know of any strong set of multistage benchmark instances.

In this paper we want to improve on this situation. We consider the natural problem of multistage shortest path (MSP). First, we propose a rich benchmark set, ranging from synthetic to real-world data, and discuss relevant aspects to ensure non-trivial instances, which is a surprisingly delicate task. Secondly, we present an explorative study on heuristic, approximate, and exact algorithms for the MSP problem from a practical point of view. Our practical findings also inform theoretical research in arguing sensible further directions. For example, based on our study we propose to focus on algorithms for multistage instances that do not rely on 2-stage oracles.

2012 ACM Subject Classification Theory of computation → Design and analysis of algorithms

Keywords and phrases Multistage Graphs, Shortest Paths, Experiments

Digital Object Identifier 10.4230/LIPIcs.SAND.2023.14

Supplementary Material *Dataset (Benchmark Instances)*: `https://tcs.uos.de/research/msp`

1 Introduction

In multistage problems, as introduced in their current form by [13, 18], we are interested in solving some problem not on a single instance, but on a sequence of instances (the *stages*) which correspond to different points in time. Such problems arise, e.g., when a certain task has to be performed multiple times at discrete points in time, but the underlying instance (in our case a graph) has received several modifications between two such time points. Thus, one typically expects two succeeding stages to be somewhat similar overall, but certainly more significantly different than being attained from a single graph operation like adding or deleting an edge. Most importantly, additionally to the original problem's objective per stage (the *stage-wise objective*), we also aim to maximize the similarity between the individual stage's solutions – the *transition quality*.

Having two distinct optimization goals at hand, one typically considers a weighted sum of both measures to allow trade-offs between the quality of the individual solutions and the similarity of those solutions [1–4, 15–17, 19]. Sometimes it is desired to guarantee optimal solutions in each stage, as first motivated in [8]; then the goal is to maximize the transition qualities by picking a suitable optimal solution (out of the set of possible optimal solutions) per stage. In either case, one may not want to yield the largest possible transition quality between two stages if this incurred an exorbitant quality decrease in other stage transitions.

Also note that our problem notion is different from many other scenarios on dynamic graphs, where the goal is to – possibly after each graph modification – update a solution as fast as possible, not (directly) caring about the specific amount of changes to the solution.

Interestingly, most polynomial-time solvable graph problems (such as shortest paths, matchings, minimum cuts, etc.) yield NP-complete problems in a multistage setting: this often already occurs when only two stages are considered, and independent on whether one restricts themselves to optimal solutions per stage or not [7, 19]. There is already some theoretical research on several variants of this problem framework; however, there is significant lack of practical evaluation. In fact, it seems that there have been no practical evaluations on any multistage graph problem so far. In this paper, we want to improve on this situation.

To this end, we consider the Multistage Shortest Path (MSP) problem, which seems to probably be the most practically relevant multistage problem. MSP was first proposed in [18] and introduced with a trade-off objective in [17]. We discuss it here in the setting where we only allow optimal solutions per stage: Given an ordered set of edge-weighted graphs (the stages) and a node pair (s, t), find a shortest s-t-path in each stage such that the subsequent paths are as similar as possible (see Section 2 for a formal definition). For example in a transportation scenario, it might be necessary but expensive to prepare each road segment before using it. Thus, we want a collection of shortest paths that allows us to reuse as many segments as possible. In a communication scenario, we prefer to use recently established channels, but not at the cost of sacrificing transfer speed. If the optimality requirement per stage appears too restricting, we point out that one can easily relax it in practice by altering the notion of what a *shortest* path is, e.g., by rounding edge weights so that all paths of reasonably similar length are considered optimal.

While the usual shortest s-t-path problem is long known to be efficiently solvable using Dijkstra's algorithm [11], MSP was shown to be NP-hard even for unweighted instances via a reduction from 3Sat [17]. Although not stated explicitly, the proof can easily be adapted to an approximation-preserving reduction from Max-2Sat which shows that, unless P = NP, MSP does not admit a PTAS nor a constant-factor approximation with factor better than 21/22 [20], even when restricted to only two stages. In [17], several similarity and dissimilarity measures were considered, and several results w.r.t. the parameterized complexity of MSP could be established. The specific formulation above, with only truly shortest paths per stage, is motivated by [8] and explicitly considered in [7] in the context of approximation algorithms.

Contribution. In this paper we improve on the state-of-the-art regarding practical algorithmics in the following two ways: First (Section 3), we propose the first rich benchmark sets for a multistage graph problem. We take special care to avoid ad-hoc generation schemes and parameterizations which, in case of MSP, would typically only yield rather trivial instances. Instead, we devise several explicit measures of reasons for triviality and actively seek schemes and parameterizations to avoid them. Secondly, we implemented and tested a set of heuristic, approximate, and exact algorithms to tackle MSP in practice (Section 4), and we report on our explorative study (Section 5). A focus of this study is to test the consistency between theoretical results and their practical realization and use it as a source for identifying new research questions.

Theoretical research suggests to improve on algorithms for the (formally already hard) 2-stage problem variant $\mathsf{MSP}|_2$, as we only know a single approximation algorithm (with non-constant approximation ratio). The multistage variants with more than two stages can reuse any 2-stage algorithm while weakening its approximation ratio only by the constant

ratio of 1/2. Interestingly, we find that in practice MSP$|_2$ problems are all rather simple to solve, but neither the known approximation nor other heuristics yield satisfactory results for general MSP. Thus, we propose that a promising step for theoretical research would be to further investigate the intricacies of the true multistage setting instead of relying on algorithms for a small constant number of stages.

The implementations will be part of the next release of the open-source (GPL) *Open Graph algorithms and Data structures Framework* [6] (`www.ogdf.net`); all benchmark instances and experimental data are available at `https://tcs.uos.de/research/msp`.

2 Definitions and Preliminaries

Given a graph G with positive edge weights $w\colon E(G) \to \mathbb{R}^+$, we encode a path $P \subseteq E(G)$ as an edge set and denote its *path length* by $\ell(P) \coloneqq \sum_{e\in P} w(e)$. Given a *query* $(s,t) \in V(G)^2$, a *shortest s-t-path* is an s-t-path with minimum path length. In contrast, we may also consider the number of *hops* (edges) $|P|$ of a path P. The *hop-distance* $h(s,t)$ is the smallest number of hops over all s-t-paths.

Let $[k] \coloneqq \{1, 2, ..., k\}$. We define a multistage graph[1] $\mathcal{G}^\tau = \langle G_i, w_i\rangle_{i\in[\tau]}$ as an ordered sequence of graphs with positive edge weights over a common node set V, i.e., $G_i = (V, E_i)$ and $w_i\colon E_i \to \mathbb{R}^+$ for all $i \in [\tau]$. Each tuple (G_i, w_i) is a *stage*, and $\mathcal{G}^\tau$ has τ stages. Observe that the weights of common edges may differ between stages.

▶ **Definition 1** (Multistage Shortest Path (MSP)). *Given a multistage graph $\mathcal{G}^\tau$ and a query $(s,t) \in V^2$, find a sequence $\mathcal{P} \coloneqq \langle P_i\rangle_{i\in[\tau]}$ of paths such that each P_i is a shortest s-t-path in G_i and the transition quality $Q(\mathcal{P}) \coloneqq \sum_{i\in[\tau-1]} |P_i \cap P_{i+1}|$ is maximized.*

If there is an upper bound T on the number of stages τ, MSP may be denoted by MSP$|_T$.

As the problem is NP-hard, we may be interested in approximate solutions. The only known approximation algorithms for MSP$|_2$ and general MSP arise as special cases of a general approximation framework [7], which in turn is a generalization of the approximation for multistage matching [8]. We will briefly summarize the algorithms later in Sections 4.2 and 4.3. For now, we may only mention that the approximation ratio is dependent on the *intertwinement* $\mu \coloneqq \max_{i\in[\tau-1]} |E_i \cap E_{i+1}|$ of the multistage graph, i.e., the maximum commonality between the edge sets of two succeeding stages. For MSP$|_2$ and MSP the algorithms yield approximation ratios of $(2\mu)^{-1/2}$ and $(8\mu)^{-1/2}$, respectively. While [7] guarantees these ratios, their tightness cannot be deduced for arbitrary subgraph problems. However, following the construction ideas of [8], it is simple to show the tightness of the ratio (up to a small constant) for MSP and MSP$|_2$.

Preprocessing. For a given query $(s,t) \in V^2$ and looking at any stage individually, we may remove all its *non-essential* edges (i.e., edges that are not in any shortest s-t-path in that stage) without altering the set of feasible solutions. We may also discard (arising) degree-0 nodes. This can be done efficiently, as given in Algorithm 1. Thus, we assume in the following that this preprocessing is always performed before running the actual algorithms. A stage G_i that has been preprocessed w.r.t. a query (s,t) has the following useful properties:

- **(i)** G_i is a DAG with unique root s and unique sink t, and
- **(ii)** for each node $v \in V(G_i)$, all paths from s to v have the same length. The same holds for all paths from v to t.

[1] This term is also sometimes used for *leveled* graphs, whose nodes are partitioned into levels, and edges join consecutive levels, see, e.g., [10]. That definition and results thereon are unrelated to our scenario.

Algorithm 1 **Preprocessing** non-essential elements in an edge-weighted graph $G=(V,E)$.

1 compute shortest path distances $d(v)$ from s to each $v \in V$ using Dijkstra's algorithm
2 remove all edges $\{(u,v) \in E \mid d(u) + w(u,v) \neq d(v)\}$
3 compute all nodes U with a path to t (via BFS from t with reversed edges)
4 remove all nodes $V \setminus U$

After the stage-wise preprocessing, both properties in particular also hold for the graph induced by the intersection $E_i \cap E_{i+1}$, for each $i \in [\tau - 1]$.

3 Benchmark Instances

Multistage problems have mostly been viewed from a theoretical perspective up to now, and there are thus no established sets of *stage-wise* temporal instances available. Furthermore, it turns out that acquiring and even generating reasonable instances is no easy feat: In our investigations, we learned that most ad-hoc generation schemes typically lead to rather trivial multistage instances. If there are only very few different (or even just one unique) shortest paths per stage, there is not much room for transition optimization; if there are several shortest paths but on very similar stages, chances are that a single solution path can be chosen throughout all stages; if the stages become too dissimilar, such that they have only few edges in common between shortest paths, it again becomes rather simple to select shortest paths that agree in terms of these edges between subsequent stages.

An adversary may argue that such issues would go away if one switches to a trade-off based objective function where the paths' lengths are allowed to deviate from the optimum in order to allow better transitions. But we disagree: Generating instances with only a trade-off based objective function in mind would easily hide the fact that such instances may become trivial for different balancing ratios between the two considered objective functions. If, however, the instances are well-designed for our scenario with truly optimal shortest paths, we expect them to be also interesting for trade-off based optimization. Thus, it is important to discuss our benchmark generating procedure in more detail than is often done otherwise. While we cannot guarantee that our instances are especially sensible for problems beyond MSP, we hope that the underlying generation methods and considerations may be useful for devising new instances for experimental studies on other multistage problems as well.

We consider four different types of benchmark instances, each with slightly different focus and motivation (see Section 3.1), and ranging from highly synthetic to real-world origins. After discussing schemes of obtaining MSP instances from underlying graphs in Section 3.2, we discuss the complexities of identifying good parameters to obtain non-trivial MSP instances in Section 3.3. Then, Section 3.4 presents the final technical parameterizations of our benchmark sets and resulting instance properties.

3.1 Rationale for the Benchmark Scenarios

As discussed in Section 2, the query-specific preprocessing of MSP instances may yield vastly smaller stages, and the preprocessed stages have a very specific structure. In particular, their size is not necessarily related to the original instance size anymore. To obtain interesting instances, a main goal is thus to generate instances with a reasonable number of shortest paths with a reasonable number of hops each, so that the preprocessing does not already essentially solve the instance.

To this end, we start with generating a highly synthetic benchmark set `grid`, which consists of long grid graphs (i.e., two-dimensional grids where one dimension is significantly smaller than the other). For a query (s, t) where s (t) is the lower left (upper right, respectively) corner of the grid, these graphs (assuming unit edge weights) already resemble preprocessed MSP instances. Further, in contrast to more quadratically-shaped grid graphs, even relatively small modifications to the graphs are likely to yield non-trivial instances.

The benchmark set `geom` contains nearest-neighbor graphs [14], generated by a random point set in the Euclidean plane. Such random graphs allow for multiple shortest paths of reasonable lengths. In contrast, other well-established randomized generation paradigms like Erdős-Renyi graphs or Barabási-Albert graphs would only yield very small diameters [5, 9]. Further, our geometric graphs have the additional benefit of (i) naturally occurring edge weights, and (ii) if one stage is generated from the previous stage by adding some random displacement to each node, they also provide a natural temporal relationship between consecutive stages.

The probably most natural application for shortest path queries is navigation in road networks. However, readily available data sets do not include temporal data suitable for MSP. Our benchmark set `hybr` thus uses real-world road networks as the underlying graph data, for which we artificially generate temporal differences between the stages. Our modification methods (see Section 3.2) are mainly motivated by this scenario, but we use the same modification methods for the previous two benchmark sets as well.

Finally, there exist real-world data sets from other applications that include time-stamped edges. Under those, we are mainly interested in email communication networks ("who wrote to whom, and when?") or human contact networks ("who was near whom, and when?"), as we can interpret these data in the context of the MSP problem: We want to quickly pass some information from source to target, while preferring interpersonal relations that have been used recently. We collect such instances in our benchmark set `real`.

3.2 Multistage Instance Generation

Modification variants. Necessarily, the stages of a multistage graph need to differ to compose a non-trivial instance. The `real` instances already have differing stages; for `grid`, `geom`, and `hybr` base instances we can use either of the following three modification schemes (applied to each stage independently) to obtain multiple differing stages. Additionally, we can also obtain differing stages for the `geom` instances by perturbing the node coordinates between stages to simulate random walks of the nodes (see Section 3.4). Keep in mind that these modifications are performed on the original graph, prior any query knowledge and thus prior to any preprocessing.

Edge deletion: Regardless of the interpretation of the instance, there is a plethora of different reasons to motivate the absence of edges in some stages. As simple examples, road closures in road networks or link failures in computer networks can lead to their temporary unavailability. Given a *modification ratio* $\lambda_E \in [0, 1)$, we remove $\lfloor \lambda_E \cdot |E_i| \rfloor$ many edges from the respective stage, chosen uniformly at random.

Node deletion: Removing arbitrarily chosen edges might (i) not alter the set of shortest paths too much and (ii) not describe all real-world scenarios too well, as the reason for an edge absence can have some local impact on surrounding edges as well. A simple method to generate local events of slightly larger impact is to remove nodes together with all incident edges – this occurs, e.g., if some road intersection is blocked, or if some server goes offline in a communication network. As above, we use a *modification ratio* $\lambda_V \in [0, 1)$ and remove $\lfloor \lambda_V \cdot |V| \rfloor$ many nodes from the respective stage, chosen uniformly at random.

Weight scaling: Some incidents (e.g., construction sites) do not render the respective node or edge completely unusable but rather increase the cost for their usage. This is typically no isolated effect but also affects the neighborhood of said graph element – the closer the proximity, the larger the effect. We model this by selecting a random node v and sorting E_i by hop-distance from v; the closer an edge is to v, the more we scale up its weight. The following precise parameters were selected subject to the discussion in Section 3.3: the weights of the $\lfloor |E_i|/8 \rfloor$ closest edges are multiplied by a factor of 4; the next $\lfloor |E_i|/4 \rfloor$ edges are multiplied by a factor of 2. Observe that if edges have exponential weights (base 2, see below) they retain this property after the scaling.

Query selection. While the `grid` instances are constructed with specific extremal queries in mind, we need to choose queries for the other three benchmark sets. To find multiple queries, each with relatively long shortest paths, we use the following randomized process. Consider some stage G_i with the least number of edges. Let $h'(v, w) \coloneqq h(v, w)$ denote the hop-distance from v to w in G_i if they are in a common component, and $h'(v, w) \coloneqq -1$ otherwise. Let $h^*(v) \coloneqq \max_{u \in V} h'(v, u)$ and $H(v) \coloneqq \{w \in V \mid h'(v, w) \geq 3/4 \cdot h^*(v)\}$ denote a set of distant nodes from v. Starting from a random node $c \in V$ in the largest connected component of G_i, we choose the source s uniformly at random from $H(c)$. The target t is in turn chosen uniformly at random from $H(s)$. If the query is not feasible for all stages, it is rejected.

3.3 Quality Criteria and Triviality Considerations

To differentiate interesting from trivial instances, multiple aspects have to align. These are specifically derived from our view on the MSP-problem but might also be generalized to classify instances for other multistage problems.

Triviality in a stage. Simple kinds of trivialities can be pinpointed to a specific stage.

Few paths: Let N denote the number of shortest s-t-paths in a single stage G_i. If $N = 0$, node t is not reachable from s and the instance could be split into two independent sub-instances before and after the i-th stage. Alternatively, if a practical application would rather incentivize similarity to the last feasible solution, we could simply remove the infeasible stage. Similarly, if $N = 1$, the shortest s-t-path is unique in stage i, and we can split the instance at this stage. The case $N \leq 1$ is trivial to check after preprocessing, since then E_i is either empty or a single path. We may discard instances with such a stage for experimental purposes. For $N \geq 2$, we consider the ratio $|E_i|/h_i(s, t)$ as a measure to estimate the non-triviality of stage G_i. Here, h_i is the hop-distance in G_i, easily computable during preprocessing.

Short paths: We may disregard instances with too small h_i, as defined above.

Triviality in a transition. Trivialities arising in transitions between stages may be harder to spot. Furthermore, even after understanding how to generate instances with reasonable stage-wise non-triviality, finding generation parameters that yield transition-wise non-trivial instances turned out to be much more fiddly and required more computational effort. E.g., to compute (or estimate) the measures below, we require optimal (or heuristic) solutions. Section 4.1 discusses a method to (in practice) acquire an optimal solution $\langle P_i \rangle_{i \in [\tau]}$ in reasonable enough time by the use of an ILP.

Small intersection: If, after preprocessing, the intersection $E' \coloneqq E_i \cap E_{i+1}$ between two consecutive stages is too small or poorly structured (e.g., if E' consists of mostly disconnected edges), all of E' might be in an optimal solution simultaneously, i.e., $E_i \cap E_{i+1} \subseteq P_i \cap P_{i+1}$. In this case, already the simplest greedy algorithm (see Section 4.2) would always find an

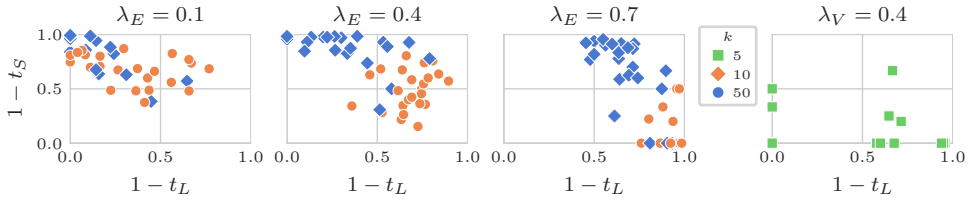


Figure 1 Triviality measures for 2-stage `geom` instances.

optimal solution. We introduce the triviality measure $t_S := \frac{|P_i \cap P_{i+1}|}{|E_i \cap E_{i+1}|}$ which compares the optimal transition quality to the intersection size. If $t_S = 1$, the transition is trivial; if t_S is close to 0, this triviality aspect plays no important role.

Large intersection: If, on the other hand, $E_i \cap E_{i+1}$ is too large, each solution edge may always also be an intersection edge, i.e., $P_i = P_i \cap E_{i+1}$ for any shortest path P_i in G_i (and similarly for P_{i+1}). We introduce the triviality measure $t_L := \frac{2 \cdot |P_i \cap P_{i+1}|}{|P_i| + |P_{i+1}|}$, comparing the optimal transition quality with the mean number of hops of the respective shortest paths. If $t_L = 1$, the transition is trivial; if t_L is close to 0, this aspect is not important.

Small transition quality: Both triviality measures t_S and t_L are not very expressive if the optimal transition quality $|P_i \cap P_{i+1}|$ is low.

Identifying non-trivial instances. The selection of the underlying graphs (and/or their generation methods) allows us to control the non-triviality within single stages in a reasonable and predictable manner. However, controlling the transition-based triviality (mainly t_S and t_L) turns out to be much more challenging. This is furthered by the fact that for a nice set of benchmarks, we would like to have similar parameterizations over all instance classes. To this end, we required multiple rounds of generating many instances starting with vastly diverse parameter selections until honing in with fine-grained parameter differences. While this may seem straight-forward on first sight, there are sometimes only very small ranges of suitable parameter values, and they may vastly drift or even disappear by slight changes to other parameters due to the interdependencies of the parameters.

We exemplarily discuss the effects on t_S and t_L by varying the modification parameters for `geom`. See Figure 1 for a visualization, where instances (points in the figure) that are trivial due to a too small (large) intersection tend to the horizontal (vertical, respectively) axis. Consider neighborhood size $k = 10$. For small λ_E, the intersection is mostly too large, causing low $1 - t_L$; for larger λ_E, the point set moves down and to the right, rendering more instances to have low $1 - t_S$. This plausible effect is evident for all instance sets, albeit with large discrepancy for sensible values of λ_E depending on, for example, k: while $\lambda_E = 0.4$ is a sensible choice for $k = 10$, $k = 50$ would benefit from a higher λ_E value. Even more so, for some instance parameterizations (e.g., $k = 5$ and $\lambda_V = 0.4$), both t_S and t_L are likely to trigger. Thus, the selected parameters are a compromise between comparability of parameter values and non-triviality w.r.t. all of the above triviality considerations.

3.4 Final Parameterization and Generation Details

After multiple rounds of investigations to identify reasonable and consistent parameterizations that yield non-trivial instances, we finally arrive at the following generation settings. Table 1 shows key figures of the generated multistage instances, given as averages over the indicated instance classes. See the appendix for more detailed tables. The full set of benchmark instances, as well as all experimental results, can be found at `https://tcs.uos.de/research/msp`.

Table 1 Instance characteristics, grouped by relevant parameters. Here, n and m (n' and m') are the mean number of nodes and edge before (after, respectively) preprocessing, always understood as the union over all stages. The ratios n'/n and m'/m thus measure the effectiveness of the preprocessing strategy. h denotes the mean hop-count of a shortest path per stage; larger numbers typically indicate higher problem difficulty. Value μ gives the mean intertwinement of the considered instances after preprocessing, which is a measure relevant to the problem's approximability (see Section 2).

`grid`	$y = 100$		$y = 200$		$y = 500$		$y = 1000$	
x	$\frac{n'}{n}$ $\frac{m'}{m}$	μ h	$\frac{n'}{n}$ $\frac{m'}{m}$	μ h	$\frac{n'}{n}$ $\frac{m'}{m}$	μ h	$\frac{n'}{n}$ $\frac{m'}{m}$	μ h
5	97.7% 97.1%	362.8 105.0	98.0% 97.3%	626.1 208.6	99.2% 98.6%	1396.5 519.6	99.4% 98.9%	2605.9 1037.4
10	89.6% 88.0%	676.5 110.2	87.8% 85.5%	962.3 214.8	90.7% 87.7%	1892.5 530.6	93.9% 90.8%	3476.2 1069.0
25	86.9% 86.0%	2356.1 123.5	77.8% 76.1%	3540.0 227.0	68.2% 64.8%	4347.4 538.6	72.0% 67.0%	6834.9 1069.2
50	91.1% 90.7%	5506.0 148.0	80.8% 80.1%	9321.3 249.0	64.8% 63.0%	14414.3 566.4	55.4% 52.1%	13940.5 1106.7

`geom`	$n = 1000$		$n = 2000$		$n = 5000$	
k	$\frac{n'}{n}$ $\frac{m'}{m}$	μ h	$\frac{n'}{n}$ $\frac{m'}{m}$	μ h	$\frac{n'}{n}$ $\frac{m'}{m}$	μ h
5	21.6% 10.5%	53.6 30.8	17.1% 8.4%	79.0 47.2	13.7% 6.9%	141.4 74.1
10	20.5% 7.7%	60.7 20.7	18.1% 6.6%	100.6 30.9	15.0% 5.7%	213.3 54.3
25	18.1% 4.7%	120.6 12.4	17.2% 4.5%	228.1 18.5	15.5% 4.0%	505.0 35.7
50	13.1% 2.3%	165.5 7.3	15.1% 3.0%	335.2 13.3	15.6% 3.2%	988.0 20.3

`hybr`	n	m	$\frac{n'}{n}$	$\frac{m'}{m}$	μ	h
`CA`	2.0M	2.8M	0.11%	0.09%	571.3	747.7
`PA`	1.1M	1.5M	0.19%	0.15%	551.4	653.6
`TX`	1.4M	1.9M	0.19%	0.15%	654.0	860.0

`real`	n	m	$\frac{n'}{n}$	$\frac{m'}{m}$	μ	h
`dnc.24`	401.8	1.2K	6.9%	5.8%	11.5	6.3
`dnc.48`	536.4	1.7K	5.5%	5.1%	15.0	5.4
`enron.168`	5.3K	12.4K	1.0%	0.8%	11.1	7.7
`enron.744`	8.9K	25.2K	0.5%	0.4%	12.4	7.4

`grid`: Long Grid Graphs. The underlying grids have $|V| = x \cdot y$ for $(x, y) \in \{5, 10, 25, 50\} \times \{100, 200, 500, 1000\}$ and unit edge weights. For each of the three modification variants we generate MSP instances with 16 stages; each stage is derived from the original underlying graph. We consider the modification ratios $\lambda_E \in \{1/5, 1/10, 1/20\}$ (for $x = 5$, $\lambda_E = 1/5$ is omitted due to generating mostly infeasible instances) and $\lambda_V = 1/20$. Overall, we generate 36 instances for each parameter combination, so we obtain 2736 `grid` instances.

`geom`: Random Nearest Neighbor Graphs. We use several parameters to generate MSP instances with 16 stages. The nodes in G_1 are $n \in \{1000, 2000, 5000\}$ randomly chosen real-valued points uniformly distributed over the unit square $[0, 1]^2$. We obtain the node position for each G_{i+1} from G_i by moving each node independently in a random direction chosen uniformly from $[-\frac{\varrho}{n}, \frac{\varrho}{n}]^2$ with $\varrho \in \{0, 1, 5\}$. In every stage, for each node we add an edge to its $k \in \{5, 10, 25, 50\}$ nearest neighbors (according to Euclidean distances). To allow for multiple shortest paths per stage, we consider two different weight functions: unit weights and exponential weights. The latter are generated by $2^{\lceil \log_2 100d \rceil}$, where d is the Euclidean distance. This mapping partitions the otherwise very diverse edge weights into buckets of (exponentially) similar weights. Due to the factor 100 (and that we have a nearest neighbor graph) we mostly observe weights in $\{1, 2, 4, 8, 16\}$.

We consider these graphs with the *edge deletion* (with $\lambda_E \in \{1/2, 1/20\}$, omitting $\lambda_E = 1/2$ for $k = 5$ and $\lambda_E = 1/20$ for $(k, n) = (50, 5000)$ due to triviality) and the *weight scaling* modifications. We do not use *node deletion* here, as these modifications (even for small

non-trivial values of λ_V) resulted in mostly trivial instances (especially many infeasible ones). However, unless $\varrho = 0$, we also consider the instances without any further modifications since the random walks of the nodes already establish differences between the stages.

Overall, we generate 4 instances for each of the 240 parameter combinations with a unique query selected according to the scheme described in Section 3.2. We obtain 960 `geom` instances overall.

`hybr`: Road Networks. We use the undirected variants of the popular real-world `roadNet` data set [25], namely the road networks of California (`CA`), Pennsylvania (`PA`), and Texas (`TX`) as three distinct underlying graphs. As the original data set does not contain any temporal information, we use the three modification variants to obtain multistage instances, with $\lambda_E \in \{1/10, 1/20, 1/100\}$ and $\lambda_V \in \{1/20, 1/100\}$. In contrast to the artificial graphs, we observe that preprocessing the underlying graph w.r.t. a query yields dramatically smaller graphs (see Table 1). Thus, we performed the stage-wise modifications *after* preprocessing (i.e., we first choose a random query as described in Section 3.2, then preprocess, modify and check feasibility), in order to guarantee that the modifications are significant within the stages. The benchmark set `hybr` consists of overall 360 multistage instances with 4 stages and a unique query each.

`real`: Communication Networks. Many real-world graph data sets with timestamped edges are email data sets [22, 23], where nodes represent people and an edge indicates a message exchange at the indicated times. We use the data sets as provided by the Konect graph collection [23], but consider edges to be undirected. Here, timestamps are given with a relatively high resolution ranging between 1 and 100 seconds, meaning that only very few events happen exactly at the same time. To generate stages with a non-trivial number of edges, we have to decrease the temporal resolution, i.e., we generate stages by accumulating all events that occur during some time window. If we choose too large time windows, the stages become too dense and yield only very short shortest paths. On the other hand, if we have too many stages, there are typically no feasible non-trivial queries possible. We thus pick time window sizes that yield interesting graphs, but restrict ourselves to 2 or 8 consecutive stages. The queries are selected as described in Section 3.2.

- `enron` [22,23]: Email communication between employees of the energy corporation *Enron*. We use time window sizes of 168 hours (a week) and 744 hours (a month). To avoid too sparse (or obviously mislabeled) data, we only consider timestamps between May 27, 1998 and Feb 04, 2004 (a span of 297 weeks).
- `dnc-email` [23]: Email communication between members of the US Democratic National Committee. We use time window sizes of 24 and 48 hours. Here, we consider timestamps between Sep 16 2013 and May 25 2016 (a span of 140 weeks).

The benchmark set `real` consists of 80 2-stage ($66 \times$ `enron`, $14 \times$ `dnc-email`) and 20 8-stage instances ($14 \times$ `enron`, $6 \times$ `dnc-email`) that are selected as those instances with the lowest triviality score, which is the sum over the values $10 \cdot \mathbb{1}[t_S = 1 \vee t_L = 1] + t_S \cdot t_L$ for the considered transitions.

We also conducted the same process on human contact data sets [12, 21], where a timestamped edge indicates a measurement of physical proximity at the given point in time. However, the resulting graphs had either a very low diameter (3 or even lower, rendering MSP essentially trivial) if the time windows were too wide, or were highly disconnected if the time windows were too narrow. There was no sweet spot between these effects and thus these instances are not included. We also note that research tells us that autonomous systems and similar networks typically experience shrinking diameters over time [24], and are thus not well-suited to yield non-trivial MSP instances.

4 Algorithms

4.1 Exact Solutions

To compute exact MSP solutions, we propose a straight-forward integer linear programming (ILP) formulation. The preprocessing routine gives us the length L_i of any shortest s-t-path in G_i, for each $i \in [\tau]$. Furthermore, it lets us define $\delta_i^+(v) \subseteq E_i$ ($\delta_i^-(v) \subseteq E_i$) as the edges that enter (leave) node v when used in a shortest s-t-path. This allows us to use a directed flow formulation for assuring the path property.

For each stage $i \in [\tau]$, the binary variable $x_i(e)$ indicates whether edge $e \in E_i$ is in P_i. Constraints (1) ensure that each P_i is an s-t-path, constraint (2) forces P_i to be of shortest length. For each transition (G_i, G_{i+1}) the (de facto binary) variable $z_i(e)$ indicates – due to constraints (3) and (4) and the objective function – whether edge $e \in E_i \cap E_{i+1}$ is in $P_i \cap P_{i+1}$. Thus, the objective function maximizes the transition quality.

$$\max \sum_{i\in[\tau-1]}\sum_{e\in E_i} z_i(e)$$

$$s.t. \sum_{e\in\delta^-(v)} x_i(e) - \sum_{e\in\delta^+(v)} x_i(e) = \mathbb{1}[v=s] - \mathbb{1}[v=t] \qquad \forall\, i \in [\tau],\ \forall\, v \in V \quad (1)$$

$$\sum_{e\in E_i} w_i(e) \cdot x_i(e) = L_i \qquad \forall\, i \in [\tau] \quad (2)$$

$$z_i(e) \le x_i(e) \qquad \forall\, i \in [\tau-1],\ \forall\, e \in E_i \cap E_{i+1} \quad (3)$$

$$z_i(e) \le x_{i+1}(e) \qquad \forall\, i \in [\tau-1],\ \forall\, e \in E_i \cap E_{i+1} \quad (4)$$

$$x_i(e) \in \{0,1\} \qquad \forall\, i \in [\tau],\ \forall\, e \in E_i \quad (5)$$

4.2 Two-Stage Algorithms

In the following, we make extensive use of the auxiliary algorithm `prefPath`(i, F). It finds, among all shortest s-t-paths in G_i, a shortest s-t-path with the maximum number of edges from F. It does so by computing Dijkstra's algorithm w.r.t. the edge weights of E_i where the weight of the edges in $F \cap E_i$ is reduced by some small ε. See [7] for details, where it is presented as the *preficiency algorithm* for MSP.

We first present some algorithms for the 2-stage problem MSP$|_2$, as these are later used as black-box algorithms for general MSP.

Greedy (`G`): Computes a shortest s-t-path $P_1 \leftarrow$ `prefPath`$(1, E_2)$ in G_1 and, favoring this path, a shortest s-t-path $P_2 \leftarrow$ `prefPath`$(2, P_1)$ in G_2.

Double Greedy (`Gd`): Calls `G` twice independently: once as described above, then with the roles of the stages interchanged. The output is the solution with larger transition quality.

Iterated Greedy (`Gi`): Computes (P_1, P_2) with `G` and then alternatingly reoptimizes $P_i \leftarrow$ `prefPath`(i, P_{3-i}) for $i = 1, 2$ until the transition quality does not improve anymore.

Approximation (`A`): This $(2\mu)^{-1/2}$-approximation algorithm from [7] iteratively computes candidate solutions (pairs of paths) and finally outputs the pair with largest transition quality. Let $Y := E_1 \cap E_2$ be the initial set of edges to be preferred. In each iteration j, a pair of paths $P_1^{(j)} \leftarrow$ `prefPath`$(1, Y)$ and $P_2^{(j)} \leftarrow$ `prefPath`$(2, P_1^{(j)})$ is computed as a new candidate solution, and we update the set of preferred edges to $Y \leftarrow Y \setminus P_1^{(j)}$. According to [7], the algorithm continues until eventually $Y = \varnothing$ (which is guaranteed to happen due to our preprocessing). Our implementation can furthermore correctly halt earlier if the current best transition quality matches the upper bound $|E_1 \cap$ `prefPath`$(2, E_1)|$.

Double Approximation (`Ad`): Similarly to how `Gd` doubles `G`, this variant calls `A` twice, the second time with the roles of the two stages interchanged. It outputs the solution with larger transition quality.

Bounded Approximation (`A5`): A variation of `A` that halts after the first 5 candidate solutions (or earlier if `A` halts earlier).

4.3 Multistage Algorithms

We consider two different polynomial-time approaches to find solutions if $\tau > 2$.

Multistage Greedy (`M-G`): After initializing $P_1 \leftarrow$ `prefPath`$(1, E_2)$, subsequent paths $P_i \leftarrow$ `prefPath`(i, P_{i-1}) are computed iteratively for $i = 2, ..., \tau$. Proceeding in the reversed direction, for each $i = \tau - 1, ..., 1$ the solution P_i is updated to `prefPath`(i, P_{i+1}). This process is repeated alternatingly front to back and back to front as long as the transition quality increases. Note that `M-G` for $\tau = 2$ coincides with `Gi`.

Multistage with black-box (`B-*`): This algorithm from [7] uses any MSP$|_2$-algorithm `*` as a black-box. The latter is executed on each consecutive pair of stages. Using a linear-time dynamic programming approach, it computes a collection of non-adjacent transitions whose transition qualities sum to the largest number. If the individual (2-stage) transitions are computed using some α-approximation, this routine yields an $\frac{\alpha}{2}$-approximation; in the case of `B-A` we thus obtain an $(8\mu)^{-1/2}$-approximation.

Improving over the description in [7] in practice, our implementation does not use arbitrary solutions for a stage that is neither optimized to the previous nor to the next stage. Instead, considering such a stage G_i, we set P_i to either `prefPath`(i, P_{i-1}) or `prefPath`(i, P_{i+1}), depending on which path yields the better transition qualities in conjunction with the solution paths of its neighboring stages (both of which are naturally fixed by the dynamic programming). We evaluate the algorithm's performance using each of the 2-stage algorithms described above as a black-box.

5 Experiments

The different algorithmic variants differ mainly in their approach for solving two-stage sub-instances. Therefore it is natural to first investigate their performance on MSP$|_2$ instances separately. Thereafter, we consider the multistage instances with $\tau > 2$.

Given some instance and some algorithm `X`, the *gap* is the ratio $(\mathsf{opt} - \mathsf{heu})/\mathsf{opt}$ where heu is the objective value computed by `X` and opt the optimal objective value.

Hard- and Software. All computations were run on an Intel Xeon Gold 6134 with 3.2 GHz and 256 GB RAM running Debian 9. We limit each run to a single thread with a 10 minute time limit. Our C++ (gcc 8.3.0) code uses OGDF Dogwood [6] as a graph algorithms library; the code will become part of the next OGDF release. We use CPLEX 20.1 as our ILP solver.

5.1 MSP$|_2$

To obtain MSP$|_2$ instances, we simply use the first two stages of every instance of `grid`, `geom` and `hybr`, as well as the two-stage instances from `real` (which, by construction, are selected to have better non-triviality than a random stage pair in the 8-stage `real` instances). See Table 2 for some average key figures on the two-stage experiments.

Nearly all two-stage algorithms are able to find solutions for all MSP$|_2$ instances within the time limit, except for `ILP`, which hits the time limit on 1.1% of the instances. In particular, due to the high success rate of `ILP` (whose few fails are restricted to very large `grid` instances), allows us to understand how often the heuristics and approximation algorithms yield optimal solutions as well. For `grid` and `geom` instances, the running times behave as one would expect on average: The greedy variants are fastest, followed by the `A` versions. The exact `ILP` is slower than the greedy approaches by up to 3 orders of magnitude (and still up to 2 orders of magnitude compared to `A` and `Ad`); only for `hybr` it is only roughly 10-fold slower

Table 2 Results for MSP$|_2$ experiments: The instances successfully solved by `ILP` yield a subset of each benchmark set for which we now know the optimal solutions. The "solved optimally" columns for `ILP` give the mean size of the respective subsets relative to the overall size of the benchmark sets. For the other algorithms, the values in the "solved optimally" columns, as well as the various "gap" columns, are then always given w.r.t. to these subsets. The columns "avg. gap" and "avg. gap (¬opt)" give the mean observed gaps to the optima, where the latter is restricted to the instances not solved to optimality by the considered algorithm. We suppress the "gap" columns for `real`, since all algorithms solved all these instances to optimality.

	time [ms]				solved optimally				avg. gap			avg. gap (¬opt)			max. gap		
	grid	geom	hybr	real	grid	geom	hybr	real	grid	geom	hybr	grid	geom	hybr	grid	geom	hybr
ILP	46787	897	2947	148	*98.9%*	*100%*	*100%*	*100%*	—	—	—	—	—	—	—	—	—
G	**20**	**3**	**185**	1	48.3%	95.0%	91.4%	100%	3.0%	0.8%	0.1%	5.9%	15.4%	0.9%	43.6%	37.5%	7.4%
Gd	41	5	316	1	56.2%	98.5%	**99.2%**	100%	1.5%	0.2%	0.0%	3.4%	11.2%	**0.3%**	23.7%	22.2%	**0.4%**
Gi	31	4	256	1	**70.4%**	96.9%	98.6%	100%	**1.0%**	0.5%	0.0%	**3.2%**	15.3%	0.7%	28.5%	33.3%	1.9%
A	972	17	361	1	48.9%	96.5%	91.4%	100%	2.6%	0.5%	0.1%	5.1%	13.0%	0.9%	27.4%	23.1%	6.0%
Ad	1913	34	660	2	56.6%	**99.0%**	**99.2%**	100%	1.4%	**0.1%**	0.0%	**3.2%**	**9.5%**	**0.3%**	**19.5%**	**16.7%**	**0.4%**
A5	46	4	346	1	48.9%	96.2%	91.4%	100%	2.6%	0.5%	0.1%	5.2%	13.1%	0.9%	27.4%	25.0%	6.0%

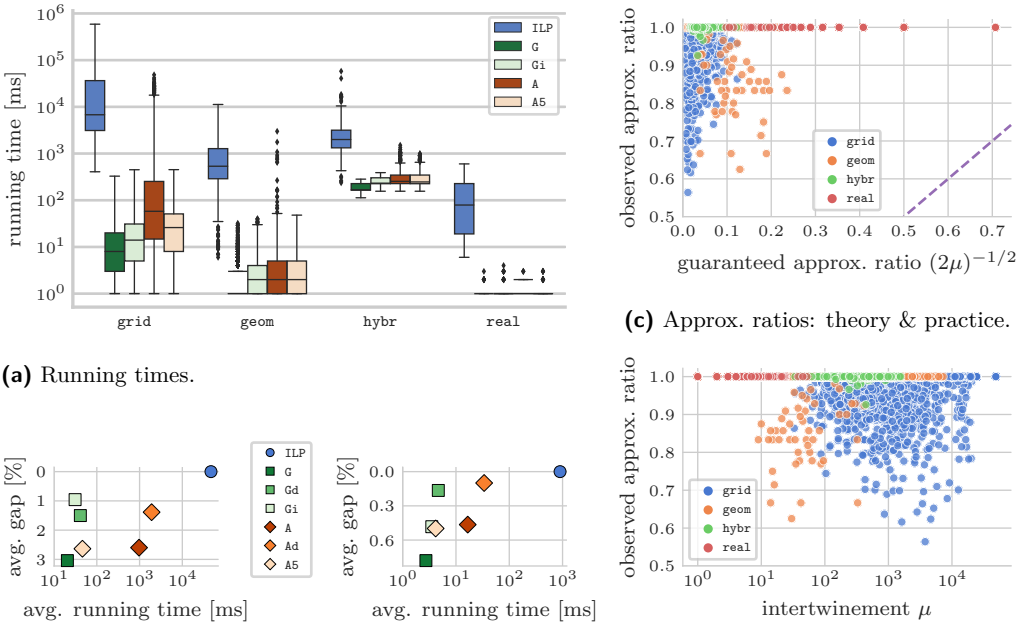
than the non-exact approaches. Naturally, `Gd` and `Ad` take about double the time of their basic counterparts. While `Gi` is slower than `G`, it is still faster than `Gd` on average: `G` and `Gd` require 2 and 4 calls to `prefPath`, respectively, but `Gi` typically terminates after the 3rd call, realizing that it cannot improve on the solution after the first two calls (i.e., the solution is identical to the one of `G`). Interestingly, `A5`'s running time is roughly comparable to that of `Gd`: it requires 2.64 iterations on average (and thus roughly 5 calls to `prefPath` on average, with a median of 2 calls), and does not suffer from outliers with a vast number of iterations as `A` does (see below). On the `hybr` benchmark set, `A` requires drastically fewer iterations than on `grid` and `geom`, and its running time becomes comparable to `A5` and thus not too far off from the greedy approaches. The running times on the `real` instances are negligibly small for all algorithms, so we refrain from analyzing them in detail.

However, as depicted in Figure 2a, the average running times do not tell the whole story. While most algorithms expose a rather predictable running time, the high variance in the running time of `A` is stunning: for many `grid` and `geom` instances, `A` spends a lot of time on later iterations that only yield candidate solutions with trivially small transition quality, but is unable to deduce that further iterations are futile.

For the following quality comparisons of the non-exact algorithms, we only consider instances with known optimal objective value (i.e. those that `ILP` could solve to proven optimality). The 2-stage `real` instances can all be solved to optimality by all algorithms. We conclude that they are, despite our best effort, still too trivial. Also the `hybr` and `geom` instances can typically be solved to optimality by most algorithms, with success rates of (clearly) above 90%. In contrast to this, the `grid` instances yield comparably hard instances for the heuristics (seemingly independent of the precise parameter choices). Note that this is also the only set where `ILP` sometimes fails to prove optimal solutions (for $(x, y) \in \{50\} \times \{500, 1000\}$). Interestingly, `Gi` finds the maximum number of optimal solutions (79.7%) overall.

Considering the average gaps, however, the difference in hardness between `grid` and `geom` seems to flip: even though many `grid` instances are not solved optimally, the observed gaps are relatively low, within one-digit percentages. In contrast to this, non-optimally solved `geom` instances typically yield gaps in the range of 10%–15% for all non-exact algorithms.

The two greedy variants `Gd` and `Gi` beat `G` w.r.t. the objective value on 20.7% and 32.5% of the instances, respectively. The average improvement over the initial greedy objective value in these cases is 30.2% and 21.6%, respectively.



(a) Running times.

(c) Approx. ratios: theory & practice.

(b) Trade-offs on `grid` (left) and `geom` (right).

(d) Observed approx. ratio vs. μ.

Figure 2 Visualizations for the MSP$|_2$ experiments. **(a)** The boxes show the median and quartiles; the whiskers extend to the farthest data point within 1.5 times the interquartile range. **(b)–(d)** We show the average gaps on all instances with known optimum; a gap g is equivalent to an observed approximation ratio of $1 - g$; the y-axes are arranged such that vertically higher data points represent solutions closer to the optimum.

For `A`, the average number of iterations (each iteration requiring two calls to `prefPath`) is 35.3. However, the actual output solution is already found after 1.2 iterations on average, generating an average computational overhead of 60.7% per instance for futile subsequent iterations. In fact, for 95.6% of the instances `A` already finds the optimal solution with the initial candidate solution (which is the same solution `G` finds). For nearly all instances (99.2%), `A` finds its output solution within the first 5 iterations, i.e., here `A5` outputs the same solution as `A`. Inversely, even if `A5` terminates earlier than `A`, this yields worse solutions only in 2.1% of those instances. Using `Ad` improves the objective value compared to `A` on 19.9% of the instances; the average improvement is 31.2% (coming at the cost of doubling the running time).

Algorithm `A` has an approximation ratio of $(2\mu)^{-1/2}$. As Figure 2c shows, `A` not only performs much better than the worst-case analysis suggests, but the correlation between the observed approximation ratio (which is $1 - g$ for gap g) and the intertwinement μ is just not very pronounced on our instances (as shown more clearly in Figure 2d). Clearly, the quite intricate instance structures necessary to yield weak approximations do typical not appear in practice (at least not in our benchmark sets).

Figure 2b shows the trade-off between solution quality (in terms of average gap over the instances solved by `ILP`) and required running time for all considered algorithms. We can conclude that the `ILP` should be preferred if running time is not an issue, and one of the two greedy approaches `Gi` or `Gd` in all other cases. The slightly better running time of `G` is typically not worth it due to the drop in quality. One could also make a case for `Ad` which, despite requiring much more time, sometimes finds slightly better solutions than the greedy variants.

Table 3 Results for MSP ($\tau > 2$) experiments: Columns are interpreted as in Table 2. Recall that the `grid`, `geom`, `hybr`, and `real` instances have 16, 16, 4, and 8 stages, respectively.

	time [ms]				solved optimally				avg. gap (¬opt)				max. gap			
	grid	geom	hybr	real	grid	geom	hybr	real	grid	geom	hybr	real	grid	geom	hybr	real
ILP	165242	6453	4202	666	*84.4%*	*100%*	*100%*	*100%*	—	—	—	—	—	—	—	—
B-G	**206**	48	**627**	13	0.0%	1.6%	10.8%	35.0%	15.6%	13.8%	3.2%	10.4%	37.6%	45.2%	14.9%	20.8%
B-Gd	397	71	1023	19	0.0%	1.7%	11.4%	35.0%	14.7%	**13.4%**	3.2%	10.4%	37.6%	45.2%	**14.5%**	20.8%
B-Gi	338	58	839	16	0.0%	1.6%	11.1%	35.0%	**14.1%**	13.6%	3.2%	10.4%	37.6%	45.2%	**14.5%**	20.8%
B-A	13922	130	1185	13	0.0%	1.5%	10.8%	35.0%	15.5%	13.6%	3.2%	10.4%	37.6%	45.2%	14.9%	20.8%
B-Ad	19096	245	2107	22	0.0%	1.6%	11.4%	35.0%	14.6%	**13.4%**	3.2%	10.4%	37.6%	45.2%	**14.5%**	20.8%
B-A5	572	68	1121	16	0.0%	1.5%	10.8%	35.0%	15.5%	13.7%	3.2%	10.4%	37.6%	45.2%	14.9%	20.8%
M-G	257	**31**	692	**8**	**0.1%**	**2.1%**	**15.3%**	0.0%	16.8%	23.5%	3.2%	23.0%	45.1%	68.8%	23.7%	42.9%

5.2 MSP

Considering the true multistage instances, i.e., $\tau > 2$, we compare `M-G` and the various variants `B-{G,Gd,Gi,A,Ad,A5}`. Some key figures are presented in Table 3.

The `ILP`'s running times increase compared to the 2-stage scenarios, but not by as much as one might expect: compared to their 2-stage counterparts, the 16-stage `grid`, 16-stage `geom`, 4-stage `hybr`, and 8-stage `real` instances require roughly 3.5x, 7.2x, 1.4x, and 4.5x more time, respectively. Thus, while `ILP` is certainly a time-wise expensive algorithm, we still can solve nearly all multistage instances to proven optimality within the time limit: it only fails on roughly 1/6 of the `grid` instances. This still allows us to investigate the ability of the non-exact algorithms to find optimal solutions.

First we may consider their running times. Observe that all `B-*` variants first run their internal $\mathsf{MSP}|_2$ algorithm for $\tau - 1$ transitions. The subsequent dynamic programming over a sequence of only $\tau - 1$ integers requires negligible time compared to the various `prefPath`-calls. Hence, the running times of these algorithms are essentially the running times observed for their internal $\mathsf{MSP}|_2$ algorithms, scaled by the number of stage transitions. `B-Ad` is the only non-exact algorithm that (on 1.3% of the `grid` instances) runs into the time limit. The running time of `M-G` is very competitive and roughly comparable with the fastest `B-*` variant, namely `B-G`.

The most interesting finding is how seldom the heuristics and the approximation approach find optimal solutions. While they all do so in most of the cases for $\mathsf{MSP}|_2$, the situation changes drastically for $\tau > 2$: We may start with discussing the `B-*` variants, as they all yield essentially the same success rates: not a single multistage `grid` instance is solved to optimality (and only mediocre 19% and 11% of `geom` and `hybr`, respectively). Even for the previously too trivial `real` instances, the algorithms find optimal solutions only for roughly a third of the 8-stage instances. The reason for this consistent picture amongst all `B-*` variants is easy to see: generally, the solution quality for the individual 2-stage sub-problems is very similar. Their common ingredient, i.e., the selection of "good" non-adjacent transitions, is to blame for the weak performance. While it is theoretically sound to simply essentially "ignore" every second transition (while still retaining an approximation guarantee), this turns out to be abysmal in practice. In fact, we can see that this worst-case scenario is even (nearly) happening for some `geom` instances: despite the fact that half of the individual transitions are essentially optimal, we observe instances with an overall gap of 45.2% – very close to the worst case of 50%. Generally, this effect overshadows the influence of the precise selection of the black-box algorithm. Even when considering the gaps yielded by the non-optimal solutions, we only see slight deviations between the variants. Interestingly, the `hybr` instances allow generally lower gaps than the other benchmark sets.

Now, one may hope that the straight-forward but reasonable sounding heuristic `M-G` may fare better, but this is also hardly the case: it finds (only) two optimal solutions on `grid` instances and is slightly more successful than `B-*` on `geom` and `hybr`. For the `real` instances, however, is fails to find any optimal solution at all. Generally over all benchmark sets, its obtained gaps are weaker than those of `B-*`. In fact, on the `grid` instances its gaps can become close to 50% and for `geom` it even achieves a solution quality only 31.2% of the optimum (a gap of 68.8%).

Overall, we can see that no non-exact algorithm comes close to the optimal solution quality obtained by `ILP`, which is thus the probably best algorithmic choice – if time is not an issue. Otherwise, we would have to recommend the use of `B-G` or `M-G`, which are comparable in quality and running time. The other more expensive MSP$|_2$ algorithms are not worth it when used within the `B-*` context on these instances.

6 Conclusion

In theory, the only known approximations for MSP$|_2$ and general MSP (`A` and `B-A`) guarantee ratios of $(2\mu)^{-1/2}$ and $(8\mu)^{-1/2} = 1/2 \cdot (2\mu)^{-1/2}$, respectively, where `B-A` uses `A` internally and only causes an additional constant ratio of $1/2$. Thus, in the hunt for better (in particular constant) approximation ratios, it seems natural to focus on stronger approximations for MSP$|_2$. However, our study shows that this is precisely *not* the interesting question when we want to obtain practically strong algorithms: MSP$|_2$ is rather simple to solve in practice, the worst-case ratios of `A` are never met, and even simple greedy heuristics find close-to-optimal solutions. In contrast, the lifting from 2 to $\tau > 2$ stages is a central weak point which undermines the algorithms' success. Also, straight-forward alternative greedy strategies (`M-G`) do not work well. We therefore propose to focus on finding true multistage approximation routines, instead of relying on simple liftings from algorithms for few stages.

In [7], a more general version of `A` is presented that is applicable to all subgraph problems that are *preficient*, which roughly speaking means that they allow a routine along the lines of `prefPath`. Thus, all `G` variants (as well as `M-G`) can also be used there, and we wonder if they perform similarly strong for such other problems as they do for MSP$|_2$ (MSP, respectively).

References

1 Evripidis Bampis, Bruno Escoffier, and Alexander Kononov. LP-based algorithms for multistage minimization problems. In Christos Kaklamanis and Asaf Levin, editors, *Approximation and Online Algorithms*, Lecture Notes in Computer Science, pages 1–15. Springer International Publishing, 2021. doi:10.1007/978-3-030-80879-2_1.

2 Evripidis Bampis, Bruno Escoffier, Michael Lampis, and Vangelis Th Paschos. Multistage matchings. In David Eppstein, editor, *16th Scandinavian Symposium and Workshops on Algorithm Theory (SWAT 2018)*, volume 101 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 7:1–7:13. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2018. ISSN: 1868-8969. doi:10.4230/LIPIcs.SWAT.2018.7.

3 Evripidis Bampis, Bruno Escoffier, Kevin Schewior, and Alexandre Teiller. Online multistage subset maximization problems. *Algorithmica*, 83(8):2374–2399, 2021-08-01. doi:10.1007/s00453-021-00834-7.

4 Evripidis Bampis, Bruno Escoffier, and Alexandre Teiller. Multistage knapsack. *Journal of Computer and System Sciences*, 126:106–118, 2022-06-01. doi:10.1016/j.jcss.2022.01.002.

5 Béla Bollobás and Oliver Riordan. The diameter of a scale-free RandomGraph. *Combinatorica*, 24(1):5–34, 2004-01-01. doi:10.1007/s00493-004-0002-2.

6 Markus Chimani, Carsten Gutwenger, Michael Jünger, Gunnar W. Klau, Karsten Klein, and Petra Mutzel. The open graph drawing framework (OGDF). In Roberto Tamassia, editor, *Handbook on graph drawing and visualization*, pages 543–569. Chapman and Hall/CRC, 2013.

7 Markus Chimani, Niklas Troost, and Tilo Wiedera. A general approach to approximate multistage subgraph problems, 2021-07-06. doi:10.48550/arXiv.2107.02581.

8 Markus Chimani, Niklas Troost, and Tilo Wiedera. Approximating multistage matching problems. *Algorithmica*, 84(8):2135–2153, 2022-08-01. doi:10.1007/s00453-022-00951-x.

9 Fan Chung and Linyuan Lu. The diameter of sparse random graphs. *Advances in Applied Mathematics*, 26(4):257–279, 2001-05-01. doi:10.1006/aama.2001.0720.

10 Huanqing Cui, Ruixue Liu, Shaohua Xu, and Chuanai Zhou. DMGA: A distributed shortest path algorithm for multistage graph. *Scientific Programming*, 2021:e6639008, 2021-06-01. Publisher: Hindawi. doi:10.1155/2021/6639008.

11 E. W. Dijkstra. A note on two problems in connexion with graphs. *Numer. Math.*, 1(1):269–271, 1959-12-01. doi:10.1007/BF01386390.

12 Nathan Eagle and Alex (Sandy) Pentland. Reality mining: sensing complex social systems. *Pers Ubiquit Comput*, 10(4):255–268, 2006-05-01. doi:10.1007/s00779-005-0046-3.

13 David Eisenstat, Claire Mathieu, and Nicolas Schabanel. Facility location in evolving metrics. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming*, Lecture Notes in Computer Science, pages 459–470. Springer, 2014. doi:10.1007/978-3-662-43951-7_39.

14 D. Eppstein, M. S. Paterson, and F. F. Yao. On nearest-neighbor graphs. *Discrete Comput Geom*, 17(3):263–282, 1997-04-01. doi:10.1007/PL00009293.

15 Till Fluschnik. A multistage view on 2-satisfiability. In Tiziana Calamoneri and Federico Corò, editors, *Algorithms and Complexity*, Lecture Notes in Computer Science, pages 231–244. Springer International Publishing, 2021. doi:10.1007/978-3-030-75242-2_16.

16 Till Fluschnik, Rolf Niedermeier, Valentin Rohm, and Philipp Zschoche. Multistage vertex cover. *Theory Comput Syst*, 66(2):454–483, 2022-04-01. doi:10.1007/s00224-022-10069-w.

17 Till Fluschnik, Rolf Niedermeier, Carsten Schubert, and Philipp Zschoche. Multistage s-t path: Confronting similarity with dissimilarity in temporal graphs. In Yixin Cao, Siu-Wing Cheng, and Minming Li, editors, *31st International Symposium on Algorithms and Computation (ISAAC 2020)*, volume 181 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 43:1–43:16. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2020. ISSN: 1868-8969. doi:10.4230/LIPIcs.ISAAC.2020.43.

18 Anupam Gupta, Kunal Talwar, and Udi Wieder. Changing bases: Multistage optimization for matroids and matchings. In Javier Esparza, Pierre Fraigniaud, Thore Husfeldt, and Elias Koutsoupias, editors, *Automata, Languages, and Programming*, Lecture Notes in Computer Science, pages 563–575. Springer, 2014. doi:10.1007/978-3-662-43948-7_47.

19 Klaus Heeger, Anne-Sophie Himmel, Frank Kammer, Rolf Niedermeier, Malte Renken, and Andrej Sajenko. Multistage graph problems on a global budget. *Theoretical Computer Science*, 868:46–64, 2021-05-08. doi:10.1016/j.tcs.2021.04.002.

20 Johan Håstad. Some optimal inapproximability results. In *Proceedings of the twenty-ninth annual ACM symposium on Theory of computing*, STOC '97, pages 1–10. Association for Computing Machinery, 1997-05-04. doi:10.1145/258533.258536.

21 Lorenzo Isella, Juliette Stehlé, Alain Barrat, Ciro Cattuto, Jean-François Pinton, and Wouter Van den Broeck. What's in a crowd? analysis of face-to-face behavioral networks. *Journal of Theoretical Biology*, 271(1):166–180, 2011-02-21. doi:10.1016/j.jtbi.2010.11.033.

22 Bryan Klimt and Yiming Yang. The enron corpus: A new dataset for email classification research. In Jean-François Boulicaut, Floriana Esposito, Fosca Giannotti, and Dino Pedreschi, editors, *Machine Learning: ECML 2004*, Lecture Notes in Computer Science, pages 217–226. Springer, 2004. doi:10.1007/978-3-540-30115-8_22.

23 Jérôme Kunegis. KONECT: the koblenz network collection. In *Proceedings of the 22nd International Conference on World Wide Web*, WWW '13 Companion, pages 1343–1350. Association for Computing Machinery, 2013-05-13. doi:10.1145/2487788.2488173.

24 Jure Leskovec, Jon Kleinberg, and Christos Faloutsos. Graphs over time: densification laws, shrinking diameters and possible explanations. In *Proceedings of the eleventh ACM SIGKDD international conference on Knowledge discovery in data mining*, KDD '05, pages 177–187. Association for Computing Machinery, 2005-08-21. doi:10.1145/1081870.1081893.

25 Jure Leskovec, Kevin J. Lang, Anirban Dasgupta, and Michael W. Mahoney. Community structure in large networks: Natural cluster sizes and the absence of large well-defined clusters, 2008-10-08. doi:10.48550/arXiv.0810.1355.

A APPENDIX

Table 4 Instance characteristics of `grid` graphs. The columns are interpreted as in Table 1. Instance sets with "—" are not generated, while " ⁄ " indicates no known optimal solution. A mean hop-count value in italics is ignoring instances with no known optimum.

`grid`	$x = 5$				$x = 10$				$x = 25$				$x = 50$			
	$\frac{n'}{n}$	$\frac{m'}{m}$	μ	h	$\frac{n'}{n}$	$\frac{m'}{m}$	μ	h	$\frac{n'}{n}$	$\frac{m'}{m}$	μ	h	$\frac{n'}{n}$	$\frac{m'}{m}$	μ	h
$n = 100$, $\lambda_E = 0.05$	96.0%	95.3%	349.8	104.5	91.4%	90.6%	911.9	108.1	96.3%	96.1%	3531.4	*123.0*	97.8%	97.7%	7771.4	⁄
$n = 100$, $\lambda_E = 0.1$	97.7%	96.8%	245.0	108.2	81.5%	79.4%	395.9	109.8	83.1%	82.3%	2206.0	*123.0*	90.5%	90.1%	5802.2	⁄
$n = 100$, $\lambda_E = 0.2$	—	—	—	—	84.6%	80.2%	198.4	117.1	60.6%	57.4%	455.1	124.5	69.7%	68.0%	2258.6	*148.0*
$n = 100$, $\lambda_V = 0.05$	97.2%	96.5%	340.5	104.5	90.6%	89.8%	824.5	108.1	95.5%	95.3%	3204.2	*123.0*	97.6%	97.6%	7029.1	⁄
$n = 100$, `scaling`	99.9%	99.8%	515.8	103.0	100.0%	100.0%	1051.7	108.0	99.2%	99.1%	2383.9	123.0	100.0%	100.0%	4668.6	148.0
$n = 200$, $\lambda_E = 0.05$	97.1%	96.3%	587.7	207.8	82.3%	80.6%	1003.3	209.4	83.3%	82.7%	5291.2	⁄	92.4%	92.2%	14171.8	⁄
$n = 200$, $\lambda_E = 0.1$	98.3%	97.2%	405.3	215.5	83.9%	80.7%	551.4	215.3	60.3%	58.5%	1885.6	*223.7*	75.4%	74.7%	8791.3	⁄
$n = 200$, $\lambda_E = 0.2$	—	—	—	—	89.3%	84.2%	306.9	231.7	60.4%	54.8%	449.1	234.3	44.6%	42.2%	1272.7	249.8
$n = 200$, $\lambda_V = 0.05$	96.5%	95.4%	502.5	208.1	83.9%	82.1%	871.8	209.7	84.8%	84.3%	4763.4	*223.0*	91.8%	91.6%	12546.6	⁄
$n = 200$, `scaling`	100.0%	100.0%	1008.9	203.0	99.8%	99.8%	2078.2	208.0	100.0%	100.0%	5310.8	223.0	100.0%	100.0%	9824.1	*248.0*
$n = 500$, $\lambda_E = 0.05$	98.7%	98.0%	1149.8	518.1	84.9%	81.9%	1451.2	517.6	57.7%	56.1%	3602.3	*524.5*	70.1%	69.5%	22019.8	⁄
$n = 500$, $\lambda_E = 0.1$	99.3%	98.6%	873.9	538.1	90.5%	86.5%	908.7	*534.4*	60.5%	56.1%	1308.3	536.3	41.7%	40.1%	3725.3	*550.3*
$n = 500$, $\lambda_E = 0.2$	—	—	—	—	93.7%	88.8%	642.1	575.1	67.4%	58.2%	582.6	571.6	43.4%	37.1%	738.7	577.1
$n = 500$, $\lambda_V = 0.05$	98.7%	98.0%	1056.4	519.1	84.1%	81.4%	1244.6	518.1	55.5%	53.8%	3009.1	*524.8*	69.0%	68.5%	18871.9	⁄
$n = 500$, `scaling`	100.0%	100.0%	2505.7	503.0	100.0%	100.0%	5215.7	508.0	99.7%	99.7%	13234.9	*523.0*	100.0%	100.0%	26715.7	⁄
$n = 1000$, $\lambda_E = 0.05$	99.0%	98.3%	1935.5	*1035.9*	90.4%	87.5%	2241.7	*1032.8*	58.4%	55.0%	3006.2	*1034.9*	41.7%	40.6%	9908.8	⁄
$n = 1000$, $\lambda_E = 0.1$	99.6%	99.0%	1574.0	*1074.9*	93.8%	90.2%	1753.5	*1065.8*	67.5%	60.3%	1628.0	*1064.5*	43.6%	39.0%	2207.7	*1070.5*
$n = 1000$, $\lambda_E = 0.2$	—	—	—	—	94.9%	89.5%	1103.1	*1146.7*	74.7%	64.0%	1019.0	*1137.1*	50.1%	40.6%	924.4	*1137.2*
$n = 1000$, $\lambda_V = 0.05$	99.3%	98.7%	1808.1	*1037.4*	90.4%	87.3%	2200.1	*1033.9*	59.7%	55.9%	2793.2	*1036.0*	41.7%	40.5%	7904.4	*1050.1*
$n = 1000$, `scaling`	99.8%	99.7%	5105.9	*1003.0*	99.8%	99.8%	10082.5	⁄	99.8%	99.7%	25728.0	⁄	99.8%	99.8%	48757.2	⁄

Table 5 Instance characteristics of `hybr` graphs. We use the same notation as in Table 4.

`hybr`	`CA`				`PA`				`TX`			
	$\frac{n'}{n}$	$\frac{m'}{m}$	μ	h	$\frac{n'}{n}$	$\frac{m'}{m}$	μ	h	$\frac{n'}{n}$	$\frac{m'}{m}$	μ	h
$\lambda_E = 0.01$	0.08%	0.06%	744.1	668.0	0.13%	0.10%	752.9	608.1	0.13%	0.10%	945.2	813.2
$\lambda_E = 0.05$	0.12%	0.09%	472.6	763.8	0.19%	0.15%	474.1	648.5	0.20%	0.16%	510.8	857.3
$\lambda_E = 0.1$	0.15%	0.12%	355.5	876.2	0.23%	0.19%	316.4	700.6	0.26%	0.20%	306.8	939.7
$\lambda_V = 0.01$	0.08%	0.06%	684.5	677.1	0.14%	0.11%	792.2	626.0	0.13%	0.10%	896.0	816.1
$\lambda_V = 0.05$	0.13%	0.11%	423.6	782.2	0.21%	0.17%	409.9	656.3	0.23%	0.18%	451.1	878.2
`scaling`	0.11%	0.08%	747.5	719.0	0.23%	0.18%	562.6	682.3	0.18%	0.14%	813.9	855.4

Table 6 Instance characteristics of `geom` graphs. We use the same notation as in Table 4. Weights are given either as unit weights (denoted as 1^x) or exponential weights (2^x).

`geom`	$k=5$				$k=10$				$k=25$				$k=50$			
	$\frac{n'}{n}$	$\frac{m'}{m}$	μ	h	$\frac{n'}{n}$	$\frac{m'}{m}$	μ	h	$\frac{n'}{n}$	$\frac{m'}{m}$	μ	h	$\frac{n'}{n}$	$\frac{m'}{m}$	μ	h
$n=1000$, 1^x, $\varrho=1$, $\lambda_E=0$	19.5%	10.3%	142.0	26.8	13.6%	6.2%	258.5	16.7	13.9%	4.5%	444.2	9.2	8.5%	1.8%	444.2	6.0
$n=1000$, 2^x, $\varrho=1$, $\lambda_E=0$	10.1%	4.9%	55.8	33.2	7.3%	2.2%	66.8	22.1	5.4%	0.7%	34.2	13.6	4.4%	0.3%	46.5	8.1
$n=1000$, 1^x, $\varrho=5$, $\lambda_E=0$	24.2%	12.0%	70.8	26.7	22.8%	10.2%	144.8	16.8	22.5%	8.7%	394.0	9.3	12.3%	2.5%	307.8	6.0
$n=1000$, 2^x, $\varrho=5$, $\lambda_E=0$	16.1%	7.3%	33.2	29.2	13.9%	4.6%	37.0	22.0	10.3%	1.9%	34.0	14.4	8.0%	0.8%	38.0	7.7
$n=1000$, 1^x, $\varrho=0$, $\lambda_E=0.2$	19.5%	11.1%	43.8	29.1	16.0%	7.0%	84.8	17.9	16.0%	4.9%	193.2	10.1	11.2%	2.3%	316.2	6.0
$n=1000$, 2^x, $\varrho=0$, $\lambda_E=0.2$	13.8%	7.4%	30.2	33.0	9.8%	3.1%	29.0	22.0	9.3%	1.6%	49.2	15.4	5.5%	0.5%	23.2	8.0
$n=1000$, 1^x, $\varrho=1$, $\lambda_E=0.2$	21.0%	11.9%	41.0	29.3	18.6%	8.4%	76.0	17.0	16.2%	5.3%	210.2	9.6	12.2%	2.4%	271.8	6.0
$n=1000$, 2^x, $\varrho=1$, $\lambda_E=0.2$	15.4%	8.1%	28.2	31.7	12.4%	4.0%	28.2	22.3	8.3%	1.3%	30.8	14.2	5.7%	0.5%	33.5	7.9
$n=1000$, 1^x, $\varrho=5$, $\lambda_E=0.2$	24.4%	11.6%	29.2	28.5	22.9%	9.8%	74.0	16.9	19.7%	6.3%	161.8	9.9	16.2%	4.2%	337.8	6.1
$n=1000$, 2^x, $\varrho=5$, $\lambda_E=0.2$	20.8%	9.1%	22.2	32.0	15.7%	4.9%	18.2	21.5	12.4%	2.1%	18.0	14.8	6.7%	0.5%	15.8	7.6
$n=1000$, 1^x, $\varrho=0$, $\lambda_E=0.5$	—	—	—	—	22.2%	10.0%	19.8	20.8	26.1%	8.1%	55.0	11.1	16.8%	4.5%	124.5	6.6
$n=1000$, 2^x, $\varrho=0$, $\lambda_E=0.5$	—	—	—	—	15.8%	5.5%	11.2	24.7	10.9%	1.9%	9.8	13.8	8.0%	0.8%	10.5	8.6
$n=1000$, 1^x, $\varrho=1$, $\lambda_E=0.5$	—	—	—	—	26.8%	10.6%	17.8	20.7	20.6%	5.6%	51.0	10.6	21.5%	4.6%	92.0	6.4
$n=1000$, 2^x, $\varrho=1$, $\lambda_E=0.5$	—	—	—	—	16.2%	5.4%	9.8	23.8	10.6%	2.0%	12.8	13.5	8.5%	0.9%	11.5	8.4
$n=1000$, 1^x, $\varrho=5$, $\lambda_E=0.5$	—	—	—	—	24.2%	9.3%	23.5	18.9	24.0%	6.9%	47.2	10.2	20.0%	4.6%	104.8	6.3
$n=1000$, 2^x, $\varrho=5$, $\lambda_E=0.5$	—	—	—	—	19.0%	5.7%	10.2	23.8	11.9%	2.2%	9.5	13.6	8.9%	0.7%	8.8	8.0
$n=1000$, 1^x, $\varrho=0$, `scaling`	23.8%	12.1%	105.2	30.0	29.4%	11.9%	142.8	19.6	26.8%	8.1%	233.8	10.8	20.2%	4.2%	444.5	6.5
$n=1000$, 2^x, $\varrho=0$, `scaling`	19.0%	8.6%	48.8	34.7	20.1%	5.9%	46.2	23.0	20.4%	2.8%	30.5	16.1	9.2%	0.7%	44.5	9.2
$n=1000$, 1^x, $\varrho=1$, `scaling`	28.3%	14.4%	83.8	28.5	34.9%	14.2%	102.2	18.7	32.5%	10.0%	356.8	10.3	25.6%	6.2%	703.8	6.7
$n=1000$, 2^x, $\varrho=1$, `scaling`	28.9%	12.7%	48.8	35.8	23.2%	6.4%	42.2	25.6	20.2%	2.8%	26.2	16.5	12.3%	1.0%	47.2	8.8
$n=1000$, 1^x, $\varrho=5$, `scaling`	31.9%	14.2%	50.5	28.9	41.6%	15.6%	66.0	18.0	37.0%	12.2%	230.2	10.1	30.5%	6.2%	191.0	6.5
$n=1000$, 2^x, $\varrho=5$, `scaling`	28.1%	10.8%	24.5	34.8	24.9%	6.7%	25.8	23.5	23.7%	3.2%	20.0	15.7	15.0%	1.2%	23.2	9.3
$n=2000$, 1^x, $\varrho=1$, $\lambda_E=0$	7.9%	4.0%	126.5	38.0	12.1%	4.9%	366.5	24.4	11.6%	3.9%	921.8	13.9	11.3%	2.9%	1411.0	8.9
$n=2000$, 2^x, $\varrho=1$, $\lambda_E=0$	8.3%	4.1%	90.2	52.4	5.2%	1.5%	86.5	33.6	5.5%	0.8%	111.2	21.3	4.9%	0.4%	124.8	16.9
$n=2000$, 1^x, $\varrho=5$, $\lambda_E=0$	17.0%	9.0%	117.0	39.4	17.4%	7.7%	251.0	24.9	22.8%	8.3%	983.8	14.7	10.0%	2.0%	559.5	9.0
$n=2000$, 2^x, $\varrho=5$, $\lambda_E=0$	13.6%	6.4%	60.5	50.2	12.9%	4.2%	64.0	34.4	10.8%	2.1%	67.5	21.2	8.9%	1.0%	92.0	17.7
$n=2000$, 1^x, $\varrho=0$, $\lambda_E=0.2$	11.8%	6.8%	45.0	39.8	18.6%	8.2%	232.5	26.4	12.3%	4.3%	270.8	14.5	14.1%	3.9%	968.2	9.5
$n=2000$, 2^x, $\varrho=0$, $\lambda_E=0.2$	13.9%	7.5%	60.5	56.1	10.8%	3.4%	64.2	35.0	7.2%	1.1%	60.8	21.0	7.1%	0.6%	55.5	17.2
$n=2000$, 1^x, $\varrho=1$, $\lambda_E=0.2$	16.2%	9.1%	53.8	41.3	16.4%	7.6%	126.2	26.2	19.7%	6.9%	365.5	14.9	17.6%	4.6%	337.8	9.5
$n=2000$, 2^x, $\varrho=1$, $\lambda_E=0.2$	12.8%	7.1%	47.5	50.2	10.7%	3.6%	59.5	36.2	7.8%	1.3%	45.8	20.8	6.1%	0.6%	35.0	16.6
$n=2000$, 1^x, $\varrho=5$, $\lambda_E=0.2$	20.2%	9.9%	42.0	40.9	19.9%	8.3%	89.2	26.4	21.0%	7.2%	371.5	14.9	17.0%	4.4%	750.5	9.0
$n=2000$, 2^x, $\varrho=5$, $\lambda_E=0.2$	15.4%	7.1%	35.8	51.3	14.1%	4.8%	32.2	35.0	10.6%	1.8%	34.5	19.7	9.1%	0.9%	40.5	15.8
$n=2000$, 1^x, $\varrho=0$, $\lambda_E=0.5$	—	—	—	—	19.6%	7.9%	26.5	28.7	21.3%	6.7%	73.5	15.9	22.6%	6.3%	204.0	9.9
$n=2000$, 2^x, $\varrho=0$, $\lambda_E=0.5$	—	—	—	—	14.7%	4.8%	13.2	33.5	11.0%	1.7%	14.0	21.0	7.8%	0.7%	16.0	16.2
$n=2000$, 1^x, $\varrho=1$, $\lambda_E=0.5$	—	—	—	—	20.4%	8.4%	31.2	27.8	18.6%	5.7%	72.0	15.0	21.8%	5.7%	245.5	10.2
$n=2000$, 2^x, $\varrho=1$, $\lambda_E=0.5$	—	—	—	—	12.6%	4.4%	17.0	33.2	10.9%	1.8%	15.5	21.8	8.0%	0.7%	16.2	16.0
$n=2000$, 1^x, $\varrho=5$, $\lambda_E=0.5$	—	—	—	—	20.0%	7.1%	28.0	27.1	20.6%	6.6%	85.8	15.0	17.8%	4.6%	210.5	9.6
$n=2000$, 2^x, $\varrho=5$, $\lambda_E=0.5$	—	—	—	—	15.1%	4.7%	13.5	32.6	11.2%	1.8%	12.8	21.3	9.1%	0.9%	11.5	14.5
$n=2000$, 1^x, $\varrho=0$, `scaling`	19.4%	10.0%	129.2	43.3	27.7%	11.0%	188.8	28.9	33.0%	9.9%	351.8	16.2	31.3%	8.2%	753.0	10.2
$n=2000$, 2^x, $\varrho=0$, `scaling`	20.5%	9.2%	85.8	55.6	19.8%	5.5%	77.5	37.1	17.6%	2.3%	159.2	24.8	13.8%	1.0%	42.8	17.6
$n=2000$, 1^x, $\varrho=1$, `scaling`	19.4%	9.9%	154.8	43.5	24.6%	9.4%	208.8	27.7	29.2%	8.9%	596.2	16.4	26.9%	6.1%	1041.5	10.8
$n=2000$, 2^x, $\varrho=1$, `scaling`	18.8%	8.9%	80.8	57.1	23.8%	6.6%	76.5	36.8	20.4%	2.8%	71.2	23.3	15.7%	1.1%	70.5	18.1
$n=2000$, 1^x, $\varrho=5$, `scaling`	30.4%	13.0%	88.0	41.9	36.2%	12.8%	119.0	28.6	35.6%	9.7%	299.8	16.2	34.3%	7.6%	355.2	10.8
$n=2000$, 2^x, $\varrho=5$, `scaling`	28.5%	11.0%	46.5	53.9	26.6%	7.0%	40.2	36.0	19.7%	2.8%	33.8	22.2	16.6%	1.4%	33.2	17.3
$n=5000$, 1^x, $\varrho=1$, $\lambda_E=0$	5.9%	3.0%	206.5	63.7	7.7%	3.3%	455.0	39.7	13.2%	4.5%	2153.2	25.1	9.5%	2.4%	2729.5	16.3
$n=5000$, 2^x, $\varrho=1$, $\lambda_E=0$	5.9%	3.0%	214.0	80.2	7.2%	2.6%	393.2	66.1	6.0%	1.1%	401.8	53.8	4.3%	0.4%	283.5	24.3
$n=5000$, 1^x, $\varrho=5$, $\lambda_E=0$	10.2%	5.3%	188.2	64.0	14.5%	6.2%	392.5	41.3	17.1%	6.2%	1263.8	24.2	14.6%	4.0%	2787.8	16.7
$n=5000$, 2^x, $\varrho=5$, $\lambda_E=0$	9.6%	4.9%	160.0	75.8	12.3%	4.9%	263.0	67.1	8.6%	1.9%	294.0	46.3	8.5%	1.1%	284.8	22.7
$n=5000$, 1^x, $\varrho=0$, $\lambda_E=0.2$	14.1%	7.8%	94.5	69.7	13.0%	5.6%	229.0	44.0	14.2%	4.9%	621.0	24.2	—	—	—	—
$n=5000$, 2^x, $\varrho=0$, $\lambda_E=0.2$	10.8%	6.2%	98.8	80.5	8.0%	3.2%	138.5	62.0	7.2%	1.4%	193.2	45.9	—	—	—	—
$n=5000$, 1^x, $\varrho=1$, $\lambda_E=0.2$	11.7%	6.3%	70.8	64.0	12.6%	5.5%	194.0	41.9	13.4%	4.6%	575.8	24.4	—	—	—	—
$n=5000$, 2^x, $\varrho=1$, $\lambda_E=0.2$	11.7%	6.4%	82.0	81.6	11.5%	4.4%	160.0	62.5	9.6%	2.0%	190.0	45.2	—	—	—	—
$n=5000$, 1^x, $\varrho=5$, $\lambda_E=0.2$	12.8%	6.5%	56.8	65.9	14.7%	6.2%	164.2	41.8	16.2%	5.9%	549.5	24.0	—	—	—	—
$n=5000$, 2^x, $\varrho=5$, $\lambda_E=0.2$	12.9%	6.4%	69.2	79.1	13.1%	4.9%	121.2	63.4	11.5%	2.4%	133.0	46.2	—	—	—	—
$n=5000$, 1^x, $\varrho=0$, $\lambda_E=0.5$	—	—	—	—	14.1%	5.7%	48.2	45.6	17.9%	5.2%	146.8	25.3	18.7%	5.0%	509.8	17.3
$n=5000$, 2^x, $\varrho=0$, $\lambda_E=0.5$	—	—	—	—	12.5%	4.6%	30.8	60.4	10.1%	2.0%	43.8	40.1	8.2%	1.0%	36.8	21.3
$n=5000$, 1^x, $\varrho=1$, $\lambda_E=0.5$	—	—	—	—	14.9%	5.8%	47.8	46.9	19.1%	5.5%	154.5	26.6	17.8%	4.2%	436.0	17.0
$n=5000$, 2^x, $\varrho=1$, $\lambda_E=0.5$	—	—	—	—	12.4%	4.4%	25.8	62.2	11.2%	2.2%	38.2	40.5	9.0%	1.0%	45.2	22.1
$n=5000$, 1^x, $\varrho=5$, $\lambda_E=0.5$	—	—	—	—	17.4%	6.1%	34.5	47.5	18.2%	5.2%	108.0	25.8	18.2%	3.9%	253.5	17.8
$n=5000$, 2^x, $\varrho=5$, $\lambda_E=0.5$	—	—	—	—	12.9%	4.4%	29.2	61.8	12.5%	2.5%	47.5	40.5	9.2%	1.0%	34.8	21.6
$n=5000$, 1^x, $\varrho=0$, `scaling`	14.1%	7.0%	219.2	69.8	19.6%	7.5%	417.0	46.1	21.1%	5.9%	1878.2	26.1	25.8%	6.2%	2086.5	18.0
$n=5000$, 2^x, $\varrho=0$, `scaling`	16.4%	8.2%	152.5	83.1	20.5%	6.8%	224.5	69.5	17.8%	2.9%	274.8	48.4	14.0%	1.3%	207.8	24.5
$n=5000$, 1^x, $\varrho=1$, `scaling`	20.2%	10.3%	210.5	69.8	22.3%	9.2%	606.8	44.3	26.0%	7.5%	595.5	26.3	25.3%	7.3%	4169.2	17.9
$n=5000$, 2^x, $\varrho=1$, `scaling`	17.8%	8.8%	186.8	86.6	19.1%	6.3%	272.5	67.1	18.9%	3.2%	328.5	48.9	16.3%	1.5%	72.2	24.5
$n=5000$, 1^x, $\varrho=5$, `scaling`	20.9%	9.3%	137.2	67.3	25.5%	9.6%	273.5	46.3	25.9%	7.2%	869.2	26.5	32.4%	8.8%	1726.5	18.3
$n=5000$, 2^x, $\varrho=5$, `scaling`	23.6%	10.6%	115.8	84.9	24.2%	7.7%	171.5	67.5	24.9%	4.4%	249.2	50.8	18.3%	1.8%	144.2	24.2