Approximation Algorithms in the Successive Hitting Set Model

Sabine Storandt[1]

Department of Computer Science, University of Freiburg (Germany)
`storandt@informatik.uni-freiburg.de`

Abstract. We introduce the successive Hitting Set model, where the set system is not given in advance but a set generator produces the sets that contain a specific element from the universe on demand. Despite incomplete knowledge about the set system, we show that several approximation algorithms for the conventional Hitting Set problem can be adopted to perform well in this model. We describe, and experimentally investigate, several scenarios where the new model is beneficial compared to the conventional one.

1 Introduction

The Hitting Set problem is a classical NP-complete problems, with applications in various areas as computational geometry [1], sensor networks [2], or route planning [3]. The unweighted Hitting Set problem (HS) is defined as follows.

Definition 1 (Hitting Set). *Given a set system $(U, \mathfrak{S})$ with U being a universe of elements and $\mathfrak{S}$ a collection of subsets of U, the Hitting Set problem demands to find a smallest subset of the universe $H \subseteq U$ such that all sets in $\mathfrak{S}$ are hit by H, i.e. $\forall S \in \mathfrak{S} : S \cap H \neq \emptyset$.*

In the weighted version, additionally a weight function $w : U \to \mathbb{R}^+$ is given. The goal is then to find the cheapest $H \subseteq U$ which hits all sets in $\mathfrak{S}$.

Both problem versions are not only NP-hard but exhibit also an inapproximability bound of $\ln(m)(1 - o(1))$ with $m = |\mathfrak{S}|$ as the dual of the Hitting Set problem, the Set Cover problem, was proven to be $\ln(n)(1-o(1))$ inapproximable with n being the number of elements in the universe [4]. Hitting Set problems are often tackled in practice with the greedy algorithm [5], as it provides an asymptotically optimal approximation guarantee of $\ln(m) + \Theta(1)$. For more refined Hitting Set problem versions, or with a priori knowledge about the set system, better approximations and custom-tailored heuristics are possible.

The main obstacle for solving Hitting Set type problems in practice is that with $\mathfrak{S} \subseteq \mathcal{P}(U)$ the number of sets might be significantly larger than the number of elements in the universe U. Therefore storing $\mathfrak{S}$ explicitly can demand enormous space, and operations on the complete set system are extremely expensive. This limits the applicability of greedy and other approximation algorithms to rather small instances.

A natural question is, whether we could solve the Hitting Set problem without having to store and investigate whole $\mathfrak{S}$ at once. Obviously, this demands that we can access certain subsets of $\mathfrak{S}$ efficiently. We formalize this idea into a new model, which we call the successive Hitting Set (SHS) model .

Definition 2 (Successive Hitting Set Model). *Given a universe of elements U, and a deterministic set generator $G : U \to \mathcal{P}(U)$. The set generator called for $u \in U$ reveals the collection of sets that contain u. The universe together with $\cup_{u \in U} G(u)$ forms the set system.*

The Hitting Set problem in the successive model remains basically unchanged. And, of course, we could just call the generator for all elements first, and then run the conventional Hitting Set algorithms to compute H. But the scope of the paper is to design (approximation) algorithms that issue calls to the generator in a way that the number of known sets (i.e. sets that have to be explicitly stored) at any point in time is significantly lower than $|\mathfrak{S}|$.

1.1 Related Work

Hitting Set problems or Set Cover problems with the set system being not fully provided a priori were tackled before in the context of an on-line model [6]. In this model, sets or elements are revealed to the algorithm in some unpredictable order and have to be handled immediately. On-line algorithms are analyzed by bounding the competitive ratio, that is the solution cost in the on-line model divided by the solution cost in the conventional off-line model. This differs significantly from our successive setting, as here the sets are not revealed by some 'adversary' but it is part of our envisioned approximation algorithms to call the set generator for elements in the universe wisely.

Furthermore, there is a wide range of heuristics for the conventional Hitting Set problem which aim at compressing the set system or avoid its explicit construction in order to be able to tackle large instances or to accelerate the computation [7–10]. Most often these heuristics are custom-tailored for certain kinds of set systems, and the focus is rather on providing good solutions in practice than on investigating theoretical approximation guarantees. We will provide successive algorithms in the following which exhibit good approximation guarantees and perform well in practice at the same time.

1.2 Contribution

– We adopt the standard greedy algorithm to work in the SHS model with an approximation guarantee of $\ln(m) + 2$.
– We show that the k-approximation for the k-Hitting Set problem via the pricing method carries over to our new model.
– We prove that for set systems with VC-dimension d, a $2dc \log(dc)$ approximation is possible in the SHS model. In the conventional model, the guarantee is $dc \log(dc)$.

– We investigate several applications where the successive model leads to a considerably reduced space consumption and/or faster computation times compared to the conventional model. Furthermore, we show that in practice our devised approximation algorithms achieve close-to-optimal solutions.

2 Preliminaries

In this paper we restrict ourselves to algorithms where the generator is only called once per element in the universe. Otherwise, every time an operation needs to be conducted in a conventional Hitting Set algorithm, the respective part of the set system is generated (if it fits in memory) and simply forgotten afterwards. But this potentially leads to a very high number of calls to the generator. Hence the time spend on set generation might dominate the total runtime, which is not what we aim for. So the paradigm in this paper is that a set once generated can only be forgotten after it was hit. This also provides us with an easy correctness prove for all our algorithms: If the generator was called for every element in the universe and the set system is empty, a feasible Hitting Set is at hand.

We use the following notation. With c we denote the size of the optimal solution H^*. We refer to the underlying set of elements for a collection of sets $\mathfrak{S}$ as $\underline{\mathfrak{S}} = \cup_{S \in \mathfrak{S}} S$. We assume the sets in $\mathfrak{S}$ to be closed under intersection, that is we cannot divide $\mathfrak{S}$ into two partitions $\mathfrak{S}_A$ and $\mathfrak{S}_B$ with $\underline{\mathfrak{S}_A} \cap \underline{\mathfrak{S}_B} = \emptyset$. This of course is only a technical restriction. If $\mathfrak{S}$ is not closed under intersection, we could define independent subproblems and solve them individually. In our algorithms, whenever a temporary set system runs empty because no sets intersect with previously chosen ones, we just call the generator for some arbitrary element (for which the generator was not already called), and proceed from there.

3 Greedy Algorithm for General Set Systems

The classical greedy algorithm for the Hitting Set problem works as follows. In every round of the algorithm, the element $u \in U$ is selected which hits most so far unhit sets. Or, in the weighted case, the element u which minimizes $w(u)/|\{S \in \mathfrak{S} : S \ni u\}|$. Then u is added to the Hitting Set H, and all newly hit sets are removed from the system. The algorithm proceeds until $\mathfrak{S}$ runs empty.

The computation of the best hitter in every round and the removal of the newly hit sets induce a complete sweep over all elements in so far unhit sets. This makes the execution of greedy quite expensive, especially in early rounds.

The greedy algorithm guarantees a $\ln(b) + \Theta(1)$ approximation with b being the size of the largest subset of $\mathfrak{S}$ that can be hit with a single element from U. As this subset potentially contains (almost) all sets from $\mathfrak{S}$, we have a $\ln(m) + \Theta(1)$ approximation with $m = |\mathfrak{S}|$.

3.1 Successive Greedy

In the SHS setting, we proceed as follows. We start with an arbitrary subset $\mathfrak{S}' \subseteq \mathfrak{S}$. (If no such set is specified, we call the generator G for an arbitrary element in U and refer to the resulting set as $\mathfrak{S}'$.) We select the best hitter for $\mathfrak{S}'$, add it to the solution H and remove all hit sets from $\mathfrak{S}'$ just like in the conventional greedy algorithm. But now, for every set that was hit, we call the set generator for all contained elements. (Of course, we never call the set generator twice for an element during the course of the algorithm, and we discard sets immediately that are already hit.) The generated sets are added to $\mathfrak{S}'$. Then the whole process is repeated. The algorithm stops after the set generator was called for every element in U, and $\mathfrak{S}'$ ran empty.

3.2 Approximation Quality

The successive greedy algorithm has very limited knowledge about the set system in every round. This is the very opposite of the way conventional greedy works, as it always selects the best hitter *globally*. Nevertheless, we will prove that the approximation guarantee of the successive greedy algorithm is quite close to the guarantee in the conventional model, as specified in the following Theorem.

Theorem 1. *Successive greedy computes a Hitting Set H with the property $|H| \leq c \cdot (\ln m + 2)$ where c denotes the optimal solution size and m being the number of sets in the complete set system.*

Proof. Let h be a hitter in the optimal solution, and $\mathfrak{S}(h)$ the collection of sets hit by h. With $s = |\mathfrak{S}(h)|$ we denote the number of sets hit by h, i.e. every set in $\mathfrak{S}(h)$ is bought at cost $w(h)/s$ (in the unweighted case $w(h) = 1$). We will argue that the total costs for $\mathfrak{S}(h)$ in the successive greedy algorithm are lower or equal to $w(h)(H_{s-1}+1)$ with H_{s-1} indicating the $(s-1)^{th}$ harmonic number.

Let h_0 be the first hitter in the course of the successive greedy algorithm which hits a set S in $\mathfrak{S}(h)$. Obviously, S is bought at cost $\leq w(h)$, as h would have been a possible choice as well. After h_0 is added to H, all sets are generated which intersect with S. Therefore, in the next round of the successive algorithm all sets $\mathfrak{S}(h) \setminus S$ are available in the temporary set system. (Of course, h_0 might hit more than one set in $\mathfrak{S}(h)$, but this would only reduce the total costs for $\mathfrak{S}(h)$.) The successive greedy algorithm could now choose h as the next hitter for $\mathfrak{S}(h)$, with a cost ratio of $w(h)/(s-1)$. So the only reason why the algorithm decides for another element h_1 is, that its ratio is even better or equal to $w(h)/(s-1)$. This ratio determines the cost for the next hit set in $\mathfrak{S}(h)$. Then we can apply the same argument recursively, providing us with a cost ratio of $\leq w(h)/(s-i)$ for hitter h_i, until $i = s-1$ and all sets in $\mathfrak{S}(h)$ are hit. Hence the total cost for all sets in $\mathfrak{S}(h)$ can be expressed as:

$$w(h) + \sum_{i=1}^{s-1} \frac{w(h)}{s-i} = w(h) + w(h) \sum_{i=1}^{s-1} \frac{1}{i} = w(h)(H_{s-1} + 1)$$

Using $H_n < \ln n + 1$, we can upper bound the costs for $\mathfrak{S}(h)$ by $w(h)(\ln(s) + 2)$. So compared to the costs of $w(h)$ for $\mathfrak{S}(h)$ in the optimal solution, we pay more by at most a factor of $\ln(s)+2$. As this is true for every hitter in the optimal solution, and $s \leq m$, successive greedy has an approximation guarantee of $\ln(m)+2$. ■

Arguing more precisely, the standard greedy algorithm exhibits an approximation guarantee of H_b with b being the size of the largest subset of $\mathfrak{S}$ that can be hit with a single element from U. This term converges to $\ln(b) + \gamma$ for growing b, with γ denoting the Euler-Mascheroni constant ($\gamma \approx 0.57721$). In the successive model, the approximation guarantee is $H_{b-1} + 1 = H_b + 1 - 1/(b-1)$.

4 Pricing Method for the k-Hitting Set Problem

We now consider the k-Hitting Set problem. Here, all sets in the collection $\mathfrak{S}$ contain at most k elements. For this special kind of Hitting Set problem, the general inapproximability bound does not apply. In fact, there exists a k-approximation algorithm which is an instance of the primal-dual method. The algorithm is called the pricing method as it assigns prices p_S to sets in the system. Initially all prices are zero, so $\forall S \in \mathfrak{S} : p_S = 0$. For every element $u \in U$, the following constraint yields $\sum_{S\in\mathfrak{S}(u)} p_S \leq w(u)$ with $\mathfrak{S}(u)$ being the collection of sets that contain u. If equality holds, the element u is called tight. The pricing method operates in rounds. In every round, a set S from $\mathfrak{S}$ is selected which contains only elements that are not tight. Then the price of the set p_S is increased as much as possible without violating any constraint. This leads to at least one of the elements in S becoming tight. The algorithm exits as soon as every set in $\mathfrak{S}$ contains some tight element. All tight elements form then the Hitting Set H.

4.1 Successive Algorithm

Again, we start with some arbitrary set $\mathfrak{S}' \subseteq \mathfrak{S}$. We select a set S from $\mathfrak{S}'$ in every round to make one of the contained elements tight. But we have to be very careful about not violating any constraints when increasing the price of S. Therefore, we maintain potential weights w' for every $u \in U$. In the beginning, we have $w'(u) = w(u)$ for all elements. Then after selecting S, we compute $\Delta = \min_{u\in S} w'(u)$. We increase the price p_S by Δ and at the same time decrease all potential weights from elements in S by Δ. At least one of those elements will have a potential weight of 0 afterwards. All elements with w' being 0 are added to H, and hit sets are removed from $\mathfrak{S}'$. Note, that it does not matter how we issue calls to the generator. We can just select some arbitrary element u in every round (for which the generator was not already called); and add the respective sets $G(u)$ to $\mathfrak{S}'$. Again, the algorithm exists as soon as the generator was called for all elements and $\mathfrak{S}'$ is empty.

4.2 Correctness and Analysis

Correctness of the successive pricing method is obvious, as the algorithm only terminates when every set in the system contains some tight element; and the tight elements coincide with the Hitting Set.

The quality analysis works similar to the conventional pricing method analysis. The only point we have to assure is not to violate any constraint. Every time we increase the price of a set, we decrease the potential weights of all contained elements. Therefore the summed prices of sets that contain a specific element are always equal to the original potential weight minus the final potential weight of the element. As potential weights never drop below 0, the summed prices of sets that contain element u are bounded by $w(u)$. Therefore all constraints are satisfied at any point in time. Accordingly, the quality analysis for the conventional pricing method can be applied, proving that the successive pricing method also has an approximation guarantee of k.

Theorem 2. *The successive pricing method returns a solution H for an instance of the k-Hitting-Set problem with $|H| \leq c \cdot k$ where c denotes the optimal solution size.*

The advantage of the successive approach is again that it requires only a very small subset of $\mathfrak{S}$ to be explicitly stored. More precisely, a *single* set being available in each round suffices for correctness and to achieve the desired approximation quality. Hence, linear space in the size of U would be enough for the pricing method to work, if our set generator can be instrumented to produce the sets per element one by one. For comparison, whole $\mathfrak{S}$ might require space in the order of $k \cdot |U|^k$.

5 Concatenated Hitting Sets

Lets assume we have some algorithm $\mathcal{A}$ for HS which provides a better approximation guarantee than the generally tight $\ln(m)$ bound by making use of characteristics of the underlying set system. A famous incarnation of such $\mathcal{A}$ is the algorithm by Brönimann and Goodrich [11] which provides for set systems with VC-dimension d a solution within $dc \log(dc)$. In this context, the VC-dimension can be regarded as a complexity measure for the set system. Low VC-dimensions are exhibited e.g. by many set systems on geometric objects [12]. We now describe a successive scheme which can exploit such algorithms $\mathcal{A}$ to find good approximate solutions while only operating on subsets of $\mathfrak{S}$.

5.1 Successive Algorithm

Like before, we start with some arbitrary subset $\mathfrak{S}_1 \subseteq \mathfrak{S}$. We apply algorithm $\mathcal{A}$ to $\mathfrak{S}_1$ conventionally. This provides us with an initial Hitting Set H_1. Then we delete the sets in $\mathfrak{S}_1$ from the system but add all sets that have a non-empty intersection with a set in $\mathfrak{S}_1$ (by calling the generator for all elements in $\underline{\mathfrak{S}_1}$). Of course, we never add already hit sets to the system at any point in time. For the newly generated set system $\mathfrak{S}_2$, we again apply $\mathcal{A}$ which leads to a second Hitting Set H_2. We repeat the process until the generator was called for every element in U and the set system is empty. The final Hitting Set returned is the union of $H_1, H_2, \cdots, H_k$.

5.2 Correctness and Approximation Quality

As every set in $\mathfrak{S}$ is generated at some point and only deleted after it was hit, correctness of the successive algorithm is obvious.

It remains to analyze the solution quality. We make the following two simple but crucial observations:

Observation 3 *The optimal solution size c_i for $\mathfrak{S}_i$ is smaller or equal to the optimal solution size $c = |H|$ for $\mathfrak{S}$ reduced to $\mathfrak{S}_i$, i.e. $c_i \leq |H \cap \underline{\mathfrak{S}_i}|$.*

Observation 4 *$\underline{\mathfrak{S}_i} \cap \underline{\mathfrak{S}_j} = \emptyset$ if $|i - j| > 1$, because all sets intersecting with $\mathfrak{S}_i$ are either already contained in $\mathfrak{S}_{i-1}$ or are created in $\mathfrak{S}_{i+1}$ and therefore hit and deleted before the construction of $\mathfrak{S}_{i+2}$.*

The second observation tells us that $\mathfrak{S}_{odd} = \mathfrak{S}_1, \mathfrak{S}_3, \cdots$ is a collection of pairwise intersection free instances, and the same is true for $\mathfrak{S}_{even} = \mathfrak{S}_2, \mathfrak{S}_4, \cdots$. According to the first observation, the optimal solution for whole $\mathfrak{S}$ requires at least as many hitters for $\mathfrak{S}_i$ as the individual optimal solution. As the union of intersection free instances can not lead to any redundant hitters, we conclude $c \geq \sum_{i=1}^{k/2} c_{2i}$ and $c \geq \sum_{i=1}^{k/2} c_{2i-1}$. In total we get $2c \geq \sum_{i=1}^{k} c_i$.

Now, we consider $\mathcal{A}$ with an approximation guarantee of $d \log(dc)$ with d denoting the VC-dimension of the set system. For any subset of $\mathfrak{S}$, the VC-dimension can not be higher than d. So we have $|H_i| \leq dc_i \log(dc_i)$. The optimal solution for any $\mathfrak{S}_i$ is smaller or equal to the global solution, i.e. $c_i \leq c$. Hence we can upper bound $|H_i|$ by $dc_i \log(dc)$. Then we can upper bound the size of the solution resulting from combining all individual H_i by:

$$\sum_{i=1}^{k} |H_i| \leq \sum_{i=1}^{k} dc_i \log(dc) = d \log(dc) \sum_{i=1}^{k} c_i \leq 2dc \log(dc)$$

The last inequality uses our lower bound for the optimal solution c as constructed above.

There are other set systems which even exhibit constant approximations [13] or PTAS [1]. The respective approximation algorithms can easily be plugged into our successive scheme and the analysis is quite similar.

Theorem 5. *For an approximation algorithm $\mathcal{A}$ which computes a Hitting Set H with the guarantee $|H| \leq f(c)$, the successive variant of $\mathcal{A}$ exhibits an approximation guarantee of $2f(c)$.*

So our successive scheme produces a solution with an approximation guarantee which is only worse by a factor of 2 compared of the original approximation guarantee. At the same time, our algorithm only requires the storage of the actual $\mathfrak{S}_i$, and operations to compute H are only performed on sub-instances.

For improving the solution quality in practice, we can apply a backwards pruning strategy. At the moment we constructed the Hitting Set H_i for $\mathfrak{S}_i$, we can check if elements in H_{i-1} become superfluous due to H_i. For that purpose, we sweep over the sets in $\mathfrak{S}_{i-1}$ that are *not* hit by H_i and only maintain their hitters in H_{i-1}.

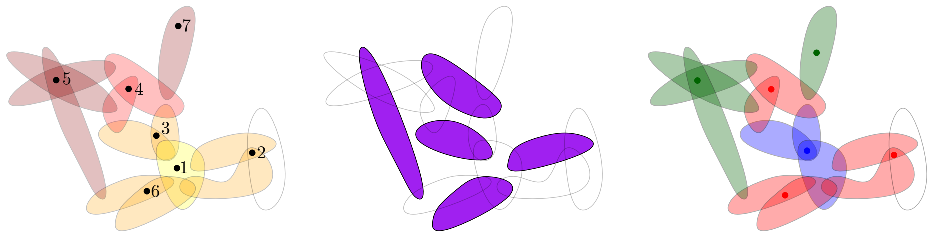

Fig. 1. Left: Illustration of the successive greedy algorithm. Hitters are indicated by black dots, the enumeration reflects the order in which they were chosen. Middle: Independent collection of sets picked by the successive pricing method. All elements in the lilac sets form the Hitting Set. Right: Illustration of the concatenation algorithm. The first instance is given by the blue sets, the second by the red sets and the third by the green sets (together with a Hitting Set per instance). The green and the blue instances are intersection-free. The colorless set on the right is not in the green instance as it is already hit by the red Hitting Set.

6 Applications

Our theoretical investigations showed that operating in the successive model leads to the same approximation guarantees in Big-O-notation than in the conventional model. But the question remaining is, if there are really applications where the intermediate sizes of the set system known to the successive approximation algorithms are considerably smaller than $|\mathfrak{S}|$.

Figure 1 illustrates all three introduced algorithms (successive greedy, successive pricing method and concatenation). For the k-Hitting Set problem we observed that a single known set suffices for the pricing method to work correctly. But for greedy and the concatenated algorithm, the existence of elements $u \in U$ with $\underline{G(u)}$ containing a significant fraction of the elements in U possibly leads to set systems with their size comparable to $\mathfrak{S}$. So in that case our successive algorithms are not advantageous.

In this section, we will describe applications where the successive model is intuitively beneficial compared to the conventional one.

Set Systems with Efficient Generators. The efficiency of our successive algorithms relies on how quick they can operate on the temporary set system as well as on how quick they can generate the next required sets. We will describe an exemplary application in the following, where efficient generators are easily available. So at the latest when the complete set system would no longer fit in memory, the successive algorithm will outperform the conventional one.

Example 1 (Hitting k-Paths or Shortest Paths). Given a graph $G(V, E)$, the objective is to hit all simple paths in G which contain at least k nodes or have a length exceeding some bound B (when additionally given a cost function $f : E \to \mathbb{R}$). The efficient construction of all paths that contain a certain vertex v can be accomplished using a breadth-first-search or Dijkstra based approach [8].

Note that the successive framework only makes sense when k or B are chosen as a small fraction of the diameter of the graph.

Incomplete Knowledge. In some applications, it might not even be possible to call the generator G a priori for every element, as necessary information might be missing. This is typically the case in AI applications, where e.g. mobile robots have to explore unknown terrain. Let, for example, the task of the robot be to physically mark every square of side length a which contains a certain amount of items in some finite area. Of course, the robot might explore the whole area first, then compute the set of all relevant squares, identify the respective marker positions, and then drive back to place them. But in the spirit of our successive scheme, it always could explore areas next that intersect with the ones just hit by driving in an a-tube around them. Then the set of squares it has to remember and that are used for computation of the next marker position(s) is smaller, and potentially the robot has to drive less of a detour to place the marker.

Solving Conventional Instances in the SHS Model. Even if the set system is explicitly available and fits in memory, it might be beneficial in terms of runtime to use the successive version of the greedy algorithm. Think of a set system where the best hitter hits only a very small fraction of all sets. Therefore the number of sets will decrease slowly in the greedy algorithm. But every round requires a complete scan over all remaining sets, so the computation gets quite expensive. In the successive algorithm, we could define $\mathfrak{S}'$ as a collection of sufficiently small sets in $\mathfrak{S}$. Then, computing the initial hitter can be made as cheap as desired. If every element can hit only a small fraction of sets in $\mathfrak{S}$, also the increase in the set system size by calling the generator is moderate.

An efficient generator for explicitly available set systems is easy to design. For example, one could store $\mathfrak{S}$ as an array of sets and keep for every element $u \in U$ a list of corresponding set indices.

7 Experiments

We implemented the standard greedy algorithm and the described successive greedy variant in C++ and evaluated them in terms of quality, space consumption and runtime. The timings were measured on a single core of an Intel i5-4300U CPU with 1.90GHz and 12GB RAM.

As example application we chose the construction of Hitting Sets on shortest paths in a graph (Example 1 in Section 6) as it is of theoretical and practical interest (see [14], [3] and [8]).

We extracted real-world road networks from OSM[1] to model the graphs. We chose networks with the number of nodes increasing from about 100,000 to 20 million. The number of edges in our test graphs is about twice the number of nodes. We demanded to find a Hitting Set for each of the graphs which hits every shortest path with a length exceeding 1000 meters. The results for

[1] `openstreetmap.org`

Table 1. Comparison of greedy and successive greedy on several benchmarks. 'k' equals 10^3. 'lb' stands for lower bound, T(ext) is time for set extraction, T(hit) for hitting set computation, T(total) or T for complete execution time. Timings are given in seconds (s), minutes (m) or hours (h).

		Greedy						Successive Greedy		
#nodes	lb	$\lvert\mathfrak{S}\rvert$	space	T(ext)	T(hit)	T(total)	$\lvert H\rvert$	max $\lvert\mathfrak{S}'\rvert$	T	$\lvert H\rvert$
100k	1,857	731k	0.17 GB	12 s	47 s	59 s	3,302	15k	21 s	4,411
500k	8,959	4,431k	1.08 GB	99 s	19 m	21 m	16,423	72k	457 s	21,313
996k	18,862	8,033k	1.98 GB	172 s	76 m	79 m	33,909	126k	18 m	41,073
6,611k	96,468	–	–	–	–	–	–	749k	10 h	246,370
21,945k	274,981	–	–	–	–	–	–	2251k	37 h	691,513

greedy and successive greedy are provided in Table 1. We observe that the greedy algorithm can only provide solutions for the instances with up to one million nodes in the graph. For larger benchmarks, the space consumption of the set system exceeds our hardware capabilities. The successive greedy algorithm on the other hand leads to results on all benchmarks. The solution quality is naturally worse compared to the classical greedy solution (about 25% on average in our experiments). To make statements about the solution quality compared to the optimum, we computed simple lower bounds along by selecting a collection of pair-wise independent sets in the system. Comparing the solutions found by successive greedy to those lower bounds (provided in Table 1, second column), we see that they are never more than a factor of 3 apart. So the approximation ratio of successive greedy is quite good in our setting.

If we compare the size of the complete set system $|\mathfrak{S}|$ to the maximum number of sets in the temporary set system (max $|\mathfrak{S}'|$) maintained by successive greedy, we observe a drastic reduction. For example, for the 996k instance, $\mathfrak{S}'$ has at most 1.5% of the size of $\mathfrak{S}$. The space consumption of $\mathfrak{S}'$ was comparable to the space consumption of the input graph for all instances. This is also reflected in the computation times. The extraction times of the set system are negligible compared to the times for the Hitting Set computation. For successive greedy the total time is always smaller than for conventional greedy, e.g. by a factor of 4.28 for the 996k instance. For larger instances this effect expectedly would be even more pronounced.

So in concurrency with out theoretical investigations, successive greedy turns out to be a useful tool to construct Hitting Sets in practice – especially on large instances.

8 Conclusions and Future Work

We introduced the successive Hitting Set model and designed algorithms that work in this model with good theoretical approximation guarantees – close to the guarantees in the conventional model. The experimental study confirmed that there are indeed applications where algorithms in the successive model lead to less space consumption and better computation times than the conventional algorithms. In future work, memory consumption could be turned into a hard constraint. We observed that the successive pricing method works if only a single set is available in every round. For successive greedy and the concatenation algorithm, the 'wavefront' of sets might become huge, though. Therefore it would be interesting to study algorithms in the successive model with the number of sets in the temporary system being restricted a priori – either instance-dependent (e.g. considering the maximal number of sets that can be hit by a single element) or completely ad hoc.

References

1. Mustafa, N.H., Ray, S.: Ptas for geometric hitting set problems via local search. In: Proceedings of the twenty-fifth annual symposium on Computational geometry, ACM (2009) 17–22
2. Hefeeda, M., Bagheri, M.: Randomized k-coverage algorithms for dense sensor networks. In: INFOCOM 2007. 26th IEEE International Conference on Computer Communications. IEEE, IEEE (2007) 2376–2380
3. Eisner, J., Funke, S.: Transit nodes-lower bounds and refined construction. In: ALENEX, SIAM (2012) 141–149
4. Feige, U.: A threshold of ln n for approximating set cover. Journal of the ACM (JACM) **45**(4) (1998) 634–652
5. Slavík, P.: A tight analysis of the greedy algorithm for set cover. In: Proceedings of the twenty-eighth annual ACM symposium on Theory of computing, ACM (1996) 435–441
6. Alon, N., Awerbuch, B., Azar, Y.: The online set cover problem. In: Proceedings of the thirty-fifth annual ACM symposium on Theory of computing, ACM (2003) 100–105
7. Vinterbo, S., Øhrn, A.: Minimal approximate hitting sets and rule templates. International Journal of approximate reasoning **25**(2) (2000) 123–143
8. Funke, S., Nusser, A., Storandt, S.: On k-path covers and their applications. Proceedings of the VLDB Endowment **7**(10) (2014)
9. Funke, S., Nusser, A., Storandt, S.: Placement of loading stations for electric vehicles: No detours necessary! In: Twenty-Eighth AAAI Conference on Artificial Intelligence. (2014)
10. Delling, D., Goldberg, A.V., Pajor, T., Werneck, R.F.: Robust distance queries on massive networks. In: Algorithms-ESA 2014. Springer (2014) 321–333
11. Brönnimann, H., Goodrich, M.T.: Almost optimal set covers in finite vc-dimension. Discrete & Computational Geometry **14**(1) (1995) 463–479
12. Matoušek, J., Seidel, R., Welzl, E.: How to net a lot with little: small epsilon-nets for disks and halfspaces. In: Proceedings of the sixth annual symposium on Computational geometry, ACM (1990) 16–22
13. Even, G., Rawitz, D., Shahar, S.M.: Hitting sets when the vc-dimension is small. Information Processing Letters **95**(2) (2005) 358–362
14. Abraham, I., Delling, D., Fiat, A., Goldberg, A.V., Werneck, R.F.: Vc-dimension and shortest path algorithms. In: Automata, Languages and Programming. Springer (2011) 690–699