# An approximation algorithm for the minimum latency set cover problem

Refael Hassin[1] and Asaf Levin[2]

[1] Department of Statistics and Operations Research, Tel-Aviv University, Tel-Aviv , Israel. hassin@post.tau.ac.il
[2] Department of Statistics, The Hebrew University, Jerusalem, Israel. levinas@mscc.huji.ac.il

**Abstract.** The input to the MINIMUM LATENCY SET COVER PROBLEM (MLSC) consists of a set of jobs and a set of tools. Each job $j$ needs a specific subset $S_j$ of the tools in order to be processed. It is possible to install a single tool in every time unit. Once the entire subset $S_j$ has been installed, job $j$ can be processed instantly. The problem is to determine an order of job installations which minimizes the weighted sum of job completion times. We show that this problem is NP-hard in the strong sense and provide an $e$-approximation algorithm. Our approximation algorithm uses a framework of approximation algorithms which were developed for the minimum latency problem.[3]

**Keywords:** Minimum sum set cover, minimum latency, approximation algorithm

## 1 Introduction

The MINIMUM LATENCY SET COVER PROBLEM (MLSC ) is defined as follows: Let $\mathcal{J} = \{J_1, J_2, \ldots, J_m\}$ be a set of *jobs* to be processed by a factory. A job $J_i$ has non-negative weight $w_i$. Let $\mathcal{T} = \{t_1, t_2, \ldots, t_n\}$ be a set of *tools*. Job $j$ is associated with a nonempty subset $S_j \subseteq \mathcal{T}$. Each time unit the factory can install a single tool. Once the entire tool subset $S_j$ has been installed, job $j$ can be processed instantly. The problem is to determine the order of tool installation so that the weighted sum of job completion times is minimized.

We rephrase MLSC as a variant of the MINIMUM SET COVER PROBLEM in the following way. Given a set of items $\mathcal{J}$ and a collection of subsets $S_1, \ldots, S_m$ of a ground set $\mathcal{T}$. We want to order the elements of $\mathcal{T}$ so that when each item

[3] As suggested by Samir Khuller there is an easy improvement of the algorithm of this paper using known 2-approximation algorithm for the minimizing weighted sum of job completion times on a single machine with precedence constraints, where the jobs in the scheduling problem consists of the set of tools and the set of MLSC-jobs, a tool-job has zero weight and unit processing time, and a MLSC-job has its weight and zero processing time, and the precedence constraints are that each MLSC-job can be processed only after all its tools are completed.

$j$ incurs a cost that equals its weight times the last time when an element of $S_j$ appears in the order. The goal is to minimize the total cost.

Feige et al. [6] considered the related problem where each job incurs a cost equal to the *first time* where an element of $S_j$ appears in the order. They proved that a greedy algorithm is a 4-approximation algorithm, and showed that unless $P = NP$ there is no polynomial time algorithm with an approximation ratio $4 - \epsilon$ where $\epsilon > 0$.

The MINIMUM LATENCY PROBLEM is defined as follows: we are given a set of $n$ points. A feasible solution is a Hamiltonian path that traverses the points. Each point $j$ (of the $n$ points) incurs a cost that equals the total length of the prefix of the path from its beginning towards the (first) appearance of $j$ in the path. The goal is to find the path that minimizes the total incurred cost. Our approximation algorithm follows similar arguments to the ones used by Goemans and Kleinberg [7] and Archer, Levin and Williamson [1] for the minimum latency problem in a metric space.

The DENSEST $k$-SUBGRAPH PROBLEM is defined as follows. We are given a graph $G = (V, E)$ where each edge $e$ has a non-negative weight $w_e$. The goal is to pick $k$ vertices $U = \{v_1, \ldots, v_k\} \subseteq V$ such that $\sum_{(u,v)\in E\cap(U\times U)} w_{(u,v)}$ is maximized. This problem is known to be NP-hard even if all weights are equal, and the current best approximation algorithm for it [5] has an approximation ratio of $O(n^{-\frac{1}{3}})$.

Given a hyper-graph $G = (V, E)$ and a subset of vertices $U \subseteq V$, a hyper-edge $e$ is induced by $U$ if $e \subseteq U$. We will use in our algorithm a new problem named the DENSEST $k$-SUB-HYPER-GRAPH PROBLEM which generalizes the densest $k$-subgraph problem to hyper-graphs where each hyper-edge $e$ contributes to the goal function if and only if it is induced by $U$. The densest $k$-sub-hyper-graph problem is at least as difficult as the densest $k$-subgraph problem. However, we are not aware of any prior results on this new variant.

**Paper preview.** In Section 2 we prove that MLSC is NP-hard in the strong sense. In Section 3 we develop a basic approximation algorithm assuming that the densest $k$-sub-hyper-graph problem can be solved in polynomial time. Afterwards, in Section 4 we show how to remove this assumption and obtain our $e$-approximation algorithm. In Section 5 we discuss bad examples for our analysis.

## 2 NP-hardness of MLSC

The following problem is known as the MINIMUM WEIGHTED SUM OF JOB COMPLETION TIMES ON A SINGLE MACHINE UNDER PRECEDENCE CONSTRAINTS WITH UNIT PROCESSING TIMES PROBLEM: given $m$ jobs $\{j_1, j_2, \ldots, j_m\}$ each has a unit processing time and a non-negative weight $w_j$, and precedence constraints between the jobs in the form of an acyclic digraph $G$. A feasible schedule must satisfy that for all $(j, k) \in A$, the machine starts to process job $k$ only after job $s$ is finished (not necessarily immediately after). The goal is to find a feasible schedule (that satisfies the precedence constraints) that minimizes the weighted sum of job completion times. In the scheduling notation this problem is denoted as $1|prec; p_j = 1|\sum w_j C_j$. This problem is known to be NP-hard in the strong sense (see [12, 13] and also [4, 10]), and there is a 2-approximation algorithm for it [8].

**Theorem 1.** *MLSC is NP-hard in the strong sense.*

*Proof.* We will describe a reduction from $1|prec; p_j = 1|\sum w_j C_j$. We are given an instance $I$ defined as $\mathcal{J} = \{j_1, j_2, \ldots, j_n\}$, such that $j_i$ has weight $w_i$, and an acyclic directed graph $G$ defining the precedence constraints. For $i = 1, \ldots, n$, let $P_i$ denote the set of all predecessors of $j_i$. We define an instance $I'$ to MLSC in the following way: Define a job $j'_i$ and a tool $t_i$ for $i = 1, \ldots, n$. Define for job $j'_i$ the job set $S'_i = \{t_i\} \bigcup_{q \in P_i} S'_q$. Finally, we assign a weight $w_i$ to $j'_i$. Denote the resulting instance of MLSC by $I'$. Since problem $1|prec; p_j = 1|\sum w_j C_j$ is NP-hard in the strong sense we can restrict ourselves to instances of $1|prec; p_j = 1|\sum w_j C_j$ in which the weights are polynomially bounded, and therefore $I'$ has polynomial size even if the numbers are represented in unary. To prove the theorem, it suffices to show that given a solution of cost $C$ to $I$ there is a solution to $I'$ of cost at most $C$, and vice versa.

First assume that $\pi$ is a feasible schedule to $I$ with cost $C$. Then, at time unit $i$ we install tool $t_{\pi(i)}$. Since $\pi$ satisfies the precedence constraints, we conclude that at time $i$ the set $S'_{\pi(i)}$ has been installed, and therefore we gain a weight of $w_{\pi(i)}$. This is exactly the weight of $\pi(i)$ such that $C_{\pi(i)} = i$ (the completion time of job $\pi(i)$ is $i$ and this term is multiplied by $w_{\pi(i)}$ in the objective function $\sum w_j C_j$). Therefore, the resulting solution costs $C$ as well.

Consider now a solution $\pi'$ to $I'$, i.e., at time $i$ we install $t_{\pi'(i)}$. W.l.o.g. we assume that prior to the $i$-th time unit we have already installed the sets $S_{i'}$ for all $i'$ such that $(i', \pi'(i)) \in G$. This assumption is w.l.o.g. because otherwise at time unit $i$ we cannot complete the processing of any job, and we can exchange the positions of the tools in $\pi'$ without additional cost. With this assumption, $\pi'$ is a feasible solution to $I$, and as in the previous case, $\pi'$ has a cost of at most $C'$. $\square$

*Remark 1.* MLSC is NP-hard even for unweighted instances. The changes that are needed in the construction is to replace a job with weight $w_j$ by a family of $w_j$ identical jobs, each with unit weight. Since MLSC is NP-hard in the strong sense the resulting instance has a polynomial size.

## 3 The Basic Approximation Algorithm

In this section, we assume that there is a polynomial time algorithm for the densest $k$-sub-hyper-graph problem. In the next section we will show how to remove this assumption. We follow similar arguments as used by Goemans and Kleinberg [7] for the minimum latency problem.

Our algorithm will make use of the values of the densest $k$-sub-hyper-graph for $k = 2, 3, \ldots, n$, in the following auxiliary hyper-graph. The vertex set is $\mathcal{T}$, for each job $j$ we will have a hyper-edge $e_j = S_j$ that is its tool set with weight $w_j$. Denote by $V_1, V_2, \dots, V_n$ the resulting vertex-sets and denote by $W_i$ the weight of hyper-edges induced by $V_i$. In other words, $W_i$ is the maximum weight of jobs that can be processed (covered) in $i$ units of time (by $i$ tools).

Given an increasing set of indices

$$j_0 = 0 < j_1 < j_2 < \cdots < j_t = n,$$

we define the *concatenated solution* as follows: for all $i = 0, \dots, t-1$, at time $\sum_{k=0}^{i} j_k$ we finished installation of the tools of the $V_{j_0} \cup V_{j_1} \cup \cdots \cup V_{j_i}$ and in the next $j_{i+1}$ time units we install the yet uninstalled tools of $V_{j_{i+1}}$ in an arbitrary order (perhaps leaving idle time until the end of this time period). A job $j$ is served at time no later than $\min\left\{\sum_{k=0}^{i} j_k \;:\; S_j \subseteq \bigcup_{k=0}^{i} V_k\right\}$. Consider the following upper bound on the cost of the concatenated solution. Suppose that the weight of jobs that are completed during the time interval $\left[\sum_{k=0}^{i-1} j_k + 1, \sum_{k=0}^{i} j_k\right]$, is $v_i$. Let $q_i = \sum_{l=0}^{i} v_l$ be the weight of jobs completed until $\sum_{k=0}^{i} j_i$. Denote by $W$ the total weight of all the jobs, i.e., $W = \sum_{j=1}^{m} w_j$.

The set $V_{j_i}$ adds at most $j_i$ to the waiting time of each of the $W - q_{i-1}$ units of weights of jobs that were not processed until time $\sum_{k=0}^{i-1} j_k$. Thus, the total cost of the concatenated solution is at most

$$\sum_{i=1}^{t} (W - q_{i-1}) \cdot j_i \leq \sum_{i=1}^{t} (W - W_{i-1}) \cdot j_i, \tag{1}$$

where the inequality follows since by definition $q_i \geq W_i$ for all $i$.

Our algorithm for approximating MLSC is as follows:

**Algorithm A:**

1. For $k = 0, 1, 2, \dots, n$, compute $V_k$, an optimal densest $k$-sub-hyper-graph solution and its value $W_k$.
2. Let $G$ be the graph on the vertex set $\{0, 1, 2, \dots, n\}$, such that, for all $i \leq j$, $G$ has an arc from $i$ to $j$ with length $(W - W_i) \cdot j$.
3. Compute a shortest $0 - n$ path in $G$. Denote its length by $\sigma$ and suppose that it goes through $j_0 = 0 < j_1 < \cdots < j_t = n$.
4. Output the concatenated solution .

The next lemma follows from (1):

**Lemma 1.** *The cost of the concatenated solution is at most $\sigma$.*

Let *opt* denote the optimal solution cost and let $\sigma$ denote the length of a shortest path in $G$.

**Theorem 2.**

$$\sigma \leq e \cdot opt.$$

*Proof.* To prove the theorem, we replace each job $j$ by $w_j$ unit weight jobs each having the same tool set $S_j$. Thus, the number of jobs is now $W$. This change clearly has no effect on *opt* or $\sigma$. Let $OPT$ be an optimal solution and denote by $l_k^*$ the time it takes $OPT$ to finish the first $k$ jobs, $k = 1, \dots, W$. Note that $1 \leq l_1^* \leq \dots \leq l_W^*$. We construct a $1 - n$ path in $G$ and compare its length to $opt = \sum_{k=1}^{W} l_k^*$.

Fix $c > 1$ and $1 \leq L_0 < c$. For $i = 1, \dots, t$ let $j_i = \lfloor L_0 c^{i-1} \rfloor$ where $t = \min\{i : L_0 c^i \geq n\}$. We may also assume w.l.o.g. that every tool is needed for some job so that the total time for the process is $n$, and therefore $j_t = n$. Consider the path $j_0 = 0, j_1, \dots, j_t = n$ in $G$. Its length is $\sum_{i=1}^{t} (W - W_{j_{i-1}}) \cdot j_i$. Since $W_t = W$,

$$\begin{aligned}
\sum_{i=1}^{t} (W - W_{j_{i-1}}) j_i &= \sum_{i=1}^{t} j_i \sum_{r=i}^{t} (W_{j_r} - W_{j_{r-1}}) \\
&= \sum_{i=1}^{t} \left[ (W_{j_i} - W_{j_{i-1}}) \cdot \sum_{r=1}^{i} j_r \right] \\
&= \sum_{k=1}^{W} \delta_k,
\end{aligned}$$

where $\delta_k = \sum_{l=1}^{i} j_l$ for $W_{j_{i-1}} < k \leq W_{j_i}$.

Let $L_k = \min\{L_0 c^i : L_0 c^i \geq l_k^*\}$.[4] By definition, $l_k^* \leq L_k$. Let $s_k$ be such that $L_k = L_0 c^{s_k}$. Therefore,

$$\delta_k = \sum_{l=1}^{s_k} j_l \leq \sum_{l=1}^{s_k} L_0 c^{l-1} = \sum_{l=1}^{s_k} \frac{L_k}{c^{s_k - l + 1}} \leq L_k + \frac{L_k}{c} + \frac{L_k}{c^2} + \dots = \frac{L_k c}{c-1},$$

where the first equation holds by definition of $\delta_k$, the first inequality holds by definition of $j_l$, the second equation holds because $L_k = L_0 c^{s_k}$.

Let $L_0 = c^U$ where $U$ is a random variable uniformly distributed over $[0, 1]$. This defines a random path whose expected length is $\sum_{k=1}^{n} E[\delta_k]$. Moreover, $E[\delta_k] \leq \frac{c}{c-1} E[L_k]$. We now compute $E[L_k]$. First, assume that $l_k^* \geq L_0$. Observe that $\frac{L_k}{l_k^*}$, is a random variable of the form $c^Y$, where $Y = \left\lceil \log_c \left( \frac{l_k^*}{L_0} \right) \right\rceil - \log_c \left( \frac{l_k^*}{L_0} \right)$ is a uniform random variable over $[0, 1]$. Hence,

$$E[L_k] = l_k^* E[c^Y] = l_k^* \int_0^1 c^x dx = l_k^* \frac{c-1}{\ln c}.$$

Even if $l_k^* < L_0$ then $L_k = L_0 \leq c$ and $E[L_k] = E[L_0] = \frac{c-1}{\ln c} \leq l_k^* \frac{c-1}{\ln c}$, where the last inequality holds because $l_k^* \geq 1$.

Thus,

$$E[\delta_k] \leq \frac{c}{\ln c} l_k^*.$$

[4] $\delta_k$ is the minimum time in our logarithmic scale that the solution defined by the path $j_0, \dots, j_t$ completes $k$ jobs, whereas $L_k$ is the time - in the same scale- it takes $OPT$ to accomplish this task.

Therefore, the expected length of our random path is at most $\frac{c}{\ln c}$ times $\sum l_k^*$. Hence, the length of a shortest path is at most $\frac{c}{\ln c}$ times $\sum l_k^*$. This value is optimized by setting $c$ to be the root of $\ln(c) - 1 = 0$, and hence $c = e \sim 2.71828$. Therefore, $\sigma \leq e \cdot opt$. □

**Corollary 1.** *Algorithm* **A** *is an e-approximation algorithm.*

## 4 The MLSC Approximation Algorithm

The results of Section 3 assumed that we are able to compute an optimal densest $k$-sub-hyper-graph for all values of $k$. In this section we remove this assumption by following the framework carried by Archer, Levin and Williamson [1] for the minimum latency problem.

For $k = 1, 2, \ldots, n$, we find either an optimal densest $k$-sub-hyper-graph or a pair of values $k_l < k < k_h$ with optimal solutions $V_{k_l}, V_{k_h}$ for the densest $k_l$-sub-hyper-graph and the densest $k_h$-sub-hyper-graph problems with costs $W_l, W_h$ (respectively) such that the following property holds: let $k = \alpha k_l + (1-\alpha)k_h$, then $a_k = \alpha W_l + (1-\alpha)W_h \geq W_k$.

We note that there are values of $k$ such that it is possible to compute a densest $k$-sub-hyper-graph in polynomial time. To make this claim precise we will show that given a parameter $\lambda > 0$ defining the cost of buying a tool, and the gain $w_j$ obtained by purchasing the subset $S_j$ (thus completing job $j$), it is possible to compute a profit maximizing set of tools. This auxiliary problem can be solved by a polynomial time algorithm for the PROVISIONING PROBLEM: Given $n$ items to choose from where item $j$ costs $c_j$, and given $m$ sets of items $S_1, S_2, \ldots, S_m$ that are known to confer special benefit; if all the items of $S_i$ are chosen then a benefit $b_i$ is gained. The goal is to maximize the net benefit, i.e., total benefit gained minus total cost of items chosen. The provisioning problem is known to be solvable in polynomial time (See [2, 14] and also [11] pages 125-127).

In fact using a single parametric min-cut procedure it is possible to compute the entire upper-envelope of the points in the graph of $W_k$ versus $k$ ([15] and see also [9] for more related results). This piecewise linear graph gives the desired $a_k$ values for all values of $k$.

For values of $k$ that for which we can compute the optimal densest $k$-sub-hyper-graph we let $a_k = W_k$, and for other values of $k$ we let $a_k = \alpha a_l + (1-\alpha)a_h \geq W_k$ and say that they corresponds to *phantom solutions* (the notion of phantom solutions is inspired by [1]). These phantom solutions are not solutions as we are not able to compute a densest $k$-sub-hyper-graph in polynomial time, however these phantom solutions provide values of $a_k$. Lemma 2 shows that there exists a shortest path in $G$ which does not use *phantom vertices*, i.e. vertices that correspond to phantom solutions, for which we are not able to compute a densest $k$-sub-hyper-graph. As a consequence, Algorithm **A** can be applied to the subgraph of $G$ induced by the vertices that correspond to non-phantom vertices, without loss of optimality.

**Lemma 2.** *There exists a shortest path in $G$ that does not use phantom vertices.*

*Proof.* We prove the lemma by showing that even when the true parameter $W_k$ of a phantom vertex $k$ is replaced by $a_k \geq W_k$, thus reducing the lengths $(W - W_k)j$ of all arcs $(k, j)$, there exists a shortest path which does not use phantom vertices.

Consider a shortest path that visits $i \to k \to j$, where $k$ is a phantom vertex with corresponding $k_l, k_h$ as defined above. Set $\gamma = \frac{a_h - a_l}{k_h - k_l}$. By definition $a_k = (1-\alpha)a_l + \gamma(k - k_l)$. By the definition of the arc lengths, the sub-path $i \to k \to j$ costs

$$\begin{aligned}(W - a_i)k + (W - a_k)j &= (W - a_i)k + (W - [(1-\alpha)a_l + \gamma(k - k_l)])j \\ &= k(W - a_i - \gamma j) + (W - (1-\alpha)a_l + \gamma k_l)j.\end{aligned}$$

This is a linear function of $k$ and it is valid for $\max\{i, k_l\} \leq k \leq \min\{j, k_h\}$. Therefore, it attains a minimum at one of the endpoints $\max\{i, k_l\}$ or $\min\{j, k_h\}$. We can either remove loops to reduce the length of the path and thus obtain a contradiction, or we reduce the number of vertices along the path that correspond to phantom vertices. Using an inductive argument we establish the lemma. □

Therefore, we can apply Corollary 1 to get the main result of this paper:

**Theorem 3.** *There is an e-approximation algorithm for the MLSC problem.*

## 5 Bad example for our analysis

Goemans and Kleinberg [7] proved that using the randomized path in their analysis does not hurt the approximation ratio (with respect to the shortest path). It follows that there are networks where the ratio between the shortest path and the optimal solution is arbitrary close to the provable approximation ratio.

In our analysis, we have different arc lengths. Therefore, the question whether our analysis is tight is open. So far we were able to construct networks (by solving large linear programs) where the ratio between the shortest 1-$n$ path to the optimal cost is approximately 2.62 for $n = 70$. This bound although monotone increasing does not approach $e$ as $n$ goes to infinity. Therefore, it might be possible to improve our analysis by using a different randomized path in the proof of the approximation ratio.

## References

1. A. Archer, A. Levin and D. P. Williamson. "Faster approximation algorithm for the minimum latency problem", *Cornell OR&IE Technical report 1362*, 2003.
2. M. L. Balinski, "On a selection problem," *Management Science,* **17**, 230-231, 1970.
3. A. Blum, P. Chalasani, D. Coppersmith, W. Pulleyblank, P. Raghavan, M. Sudan. "The minimum latency problem", *Proceeding of the 26th ACM Symposium on the Theory of Computing*, 163-171, 1994.
4. P. Brucker, "Scheduling algorithms," Springer-Verlag, Berlin, 2004.

5. U. Feige, D. Peleg and G. Kortsarz, “The dense $k$-subgraph Problem,” *Algorithmica*, **29** 410-421, 2001.
6. U. Feige, L. Lovász and P. Tetali, “Approximating min sum set cover,” *Algorithmica*, **40**, 219 - 234, 2004.
7. M. X. Goemans and J. Kleinberg. “An improved approximation ratio for the minimum latency problem”, *Mathematical Programming*, 82:111-124, 1998.
8. L. A. Hall, A. S. Schulz, D. B. Shmoys and J. Wein, "Scheduling to minimize average completion time: off-line and on-line approximation algorithms," *Mathematics of Operations Research*, **22**, 513-544, 1997.
9. D. S. Hochbaum, “Economically preferred facilities locations with networking effect,” manuscript, 2004.
10. B. J. Lageweg, J. K. Lenstra, E. L. Lawler and A. H. G. Rinnooy Kan, “Computer-aided complexity classification of combinatorial problems,” *Communications of the ACM*, **25**, 817-822, 1982.
11. E. L. Lawler, *Combinatorial Optimization: Networks and Matroids,* Holt, Rinehart & Winston, New-York 1976.
12. E. L. Lawler, “Sequencing jobs to minimize total weighted completion time subject to precedence constraints”, *Annals of Discrete Mathematics*, **2**, 75-90, 1978.
13. J.K. Lenstra and A. H. G. Rinnooy Kan, “Complexity of scheduling under precedence constraints,” *Operations Research*, **26**, 22-35, 1978.
14. J. Rhys, “Shared fixed cost and network flows,” *Management Science,* **17**, 200-207, 1970.
15. D. D. Witzgall and R. E. Saunders, “Electronic mail and the locator’s dilemma,” In *Applications of Discrete Mathematics*, R.D. Ringeisen and F.S. Roberts eds. SIAM 65-84, 1988.