Minimizing the Cost of Mine Selection Via Sensor Networks

Changlei Liu and Guohong Cao
Department of Computer Science & Engineering
The Pennsylvania State University
E-mail: {chaliu, gcao}@cse.psu.edu

***Abstract*— In this paper, we study sensor enabled landmine networks by formulating a minimum-cost mine selection problem. The problem arises in a target defence scenario, where the objective is to destroy the intruding targets using the minimum-cost pre-deployed mines. Due to the problem complexity, we first transform it using a novel bucket-tub model, and then propose several approximation algorithms. Among them, it is shown that the layering algorithm can achieve an approximation ratio of $\alpha \cdot f$, where $\alpha \geq 1$ is the tunable relaxation factor and f is the maximum number of mines that a target is associated with, and that the greedy algorithm has an approximation ratio of $\sum_j R_j$, where R_j is the coefficient in the related integer program. We also present a localized greedy algorithm which is shown to produce the same solution set as the global greedy algorithm. Theoretical analysis and extensive simulations demonstrate the effectiveness of the proposed algorithms.**

I. INTRODUCTION

Traditional landmines are commonly triggered by the pressure of the moving target (e.g., tank, vehicle, personnel) that steps on it. However, there are cases where the ground surface is not suitable to bury the mine or the mine needs to be triggered by the target within a certain distance. In these cases, the pressure-triggered mine can no longer be used or bring optimal performance. In addition, the traditional landmines lack of self-destruction capability and may linger as a threat for a long time, causing the postwar disposal issue. Therefore, many ongoing efforts have focused on developing the off-route mines that could be triggered by means such as sound, magnetism and vibration, so that the mine could detonate even when it is not touched. As far as we know, the work in [1] is the only published effort to achieve this goal by integrating the latest sensor technology [2] into the landmine design (so called smart-mine). In agreement with [1], we believe that the marriage with sensor technology could bring new opportunities and even revolutionize the entire mine industry. While [1] mainly focuses on the single sensor-enabled mine design, in this paper, we take one step further and investigate the networking opportunities that the sensor technology can bring to the next generation landmine.

Research in the impact engineering [3], [4], [5], [6] shows that the destructive effect of a mine explosion on a target depends on many factors such as the type/model of the target and mine, the distance between the target and mine, etc. In this regard, the sensor technology fills the gap between the mine industry and impact engineering by providing information crucial for decision making. With the embedded sensor, a mine is able to detect, classify the target, or even measure the distance from the target. By forming a smart-mine network, different mines can further exchange status information with each other, thus reaching agreement on a globally efficient strategy. Fig. 1(a) shows a small smart-mine network of 2 targets and 5 smart-mines, where the link indicates whether the target is within the explosion range of the mine. Based on the sensed distance information, each mine can estimate the blast impact on its neighboring target. Then different mines could exchange the information with one another and collaboratively decide who need to be triggered. For example, if mine 1 is close enough to disable both targets, the other mines do not have to be triggered; otherwise, more mines need to participate.

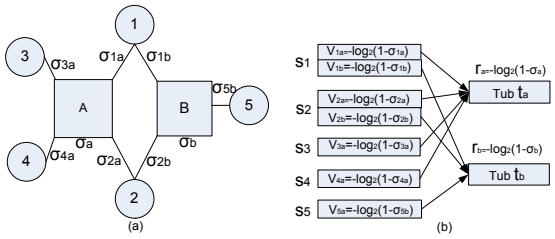


Fig. 1. (a) A small smart-mine network, with the square denoting the targets, and the circle denoting the smart-mine. (b) the corresponding bucket-tub model, with the bucket set denoting the mine and the empty tub denoting the target.

In this paper, we formulate the detonable mine selection problem based on two observations. First, as a single mine may not be powerful enough to destroy a single target, multiple mines may be used. Second, if several targets are within the explosion scope of a mine, one explosion could destroy multiple targets at the same time. Therefore, we need to consider the effects of multiple explosions on one target as well as the effect of single explosion on multiple targets. Our objective is to minimize the total cost of mines (i.e., the number of mines if each mine has the same cost) subject to the constraint that all the targets should be destroyed with a predefined probability. For example in Fig. 1(a), the objective is to select a minimum number of mines to destroy the two targets with a given probability. To resolve this problem, there are several challenges. For example, how to quantify the destructive effect of the mine explosion on a target, how to set the criteria based on which different mines are picked,

This work was supported in part by the National Science Foundation under grant CNS-0519460.

and how to find a way to let different smart-mines to negotiate with each other.

To address these challenges, we employ the results in impact engineering and calculate the collaborative effect of multiple explosions. Due to the complexity of the formulation, we transform the problem using a novel bucket-tub model and prove that it is NP-hard. Based on the new problem formulation, we propose two classes of approximation algorithms, i.e., greedy algorithm and layering algorithm. It is proved that the layering algorithm can achieve an approximation ratio of $\alpha \cdot f$, where $\alpha \geq 1$ is the tunable relaxation factor and f is the maximum number of mines that a target is associated with. In other words, the cost of the layering algorithm is bounded by $\alpha \cdot f$ times the minimum cost of the optimal solution. The greedy algorithm is shown to have an approximation ratio of $\sum_j R_j$, where R_j is the coefficient in the related integer program. To facilitate different mines to negotiate with each other in a distributed manner, we also present a local greedy algorithm, which produces the same solution set as the global greedy algorithm. By using parallel operations, the algorithm execution time can be significantly reduced.

The rest of the paper is organized as follows. Section II overviews the related work. Section III formulates the problem and transforms it using a novel bucket-tub model. Section IV proposes the greedy algorithm and layering algorithm, and proves their approximation ratios. Section V shows how to implement the greedy algorithm in a distributed manner. Performance evaluations are done in Section VI and Section VII concludes the paper.

II. RELATED WORK

The recent advance in wireless sensor networks has brought new challenges and horizons to many fields. In [1], [7], there are some ongoing efforts to integrate the sensor into the military weapon system, with the purpose to improve the military performance and reduce the civilian casualties. In [7], a sensor network-based mobile countersniper system is introduced. Their purpose is to use soldier-wearable networked sensors to localize the shooter and classify the weapons. In [1], The authors discuss the feasibility of sensor-enabled landmine, so called smart-mine. Compared with the traditional landmine, the smart-mine has distinguished features such as the target discernment, ability to talk to the neighboring mine, and self destruction after the war ceases. Different from [1], which mainly focus on the feasible design of a single smart-mine, we investigate the advantage of forming a sensor network and study how to minimize the landmine cost.

In the field of impact engineering, numerous research has been carried out to study the effect of the buried mine explosion. Depending on the types of targets, these works fall into several categories. In [5], [6], the authors study the blast injury on human. It is shown that the distance between a person and the bomb is one of the major factors in determining the survival probability. On the other hand, predicting the blast effect on the armed vehicles or tanks has also received lots of attentions in recent years. Many mathematical and experimental results about the deformation or failure of a calmed plate subject to the explosive loading are reported [3], [4]. There are a variety of models and each model uses different criteria to describe the failure modes, mental properties, soil condition, mine characteristics, rupture strain, etc. In all the models, the standoff distance from the explosion point plays an important role in determining the blast impact on the vehicles or tanks.

In this paper, we make the first attempt to fill the gap between the mine industry and the impact engineering by formulating the detonable mine selection problem. With the explosion scope mapped to the sensing range, our problem looks alike the sensor coverage problem, whose objective is to select the minimum number of sensors to cover the whole field [8], [9], [10], [11]. However, the existing techniques cannot be simply applied, because most work on coverage assumes a disk-model, where a point is thought 100% covered if it is within the sensing disk of a sensor. The simplified assumption makes the problem tractable but may not be realistic. There is also some recent work based on the probability coverage model [12]. However, in their work it is assumed that the sensors are densely deployed (i.e., each point is covered by at least three sensors), and the coverage probability of each point can be pre-computed independent of the targets. Besides the economic issue of deploying a smart-mine network with high density, it is infeasible to compute the target's survival probability before the targets enter the field. Thus, decision has to be made on the fly after a intruder is detected, classified, and localized.

III. PROBLEM FORMULATION

We consider a defense scenario where m intruders (targets) enter a field with n smart-mines. Each smart-mine relies on the embedded sensors to cooperate with each other to detect, classify, and localize the targets. By employing the results in impact engineering, each mine is assumed to be able to estimate the consequence of explosion on the intruding target. Following some literatures [5], [6], we use the metric of *failure probability* to quantitatively measure the blast impact. The failure probability reflects the chance that the target fails to function properly after the explosion due to the consequence such as personnel casualty, flat tire, or the tearing of the plate. For example, Fig. 2(a) shows the ESTC outdoor blast model [5], where the failure probability (P) of the intruder is a function of range and quantity of explosives. Fig. 2(b) further depicts the relationship between the failure probability and standoff distance, when $Q = 8$ kilograms.

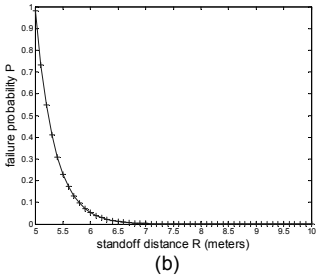
There are also other models available based on the metric of deformation factor, which describes the concrete degree of the plate deformation that the explosive force has caused to the metal-made targets such as tank and armored vehicle. We leave its study to the future work.

We assume the distance between each target and mine is known, e.g., via GPS or other localization schemes [13]. With the known distance, the failure probability of the target can be estimated, e.g., via the mathematical model [5] or experiential model [6]. With these assumptions, the *detonable mine selection problem (minMS)* is formalized as follows.

$$P = \frac{e^{\frac{[-5.785 \times (\frac{R}{Q^{\frac{1}{3}}}) + 19.047]}{Q^{\frac{1}{3}}}}}{100}$$

where R is standoff distance and Q is the quantity of explosives. The formula is valid if $2.5 < R/Q^{\frac{1}{3}} < 5.3$.

(a)



(b)

Fig. 2. the ESTC outdoor blast model ($Q = 8$ kilogram)

Given a smart-mine network with n mines and m targets, a failure based blast model, a set of probabilities $\sigma_j, j = 1...m$, we want to select the minimum-cost subset of mines such that each target j can be destroyed with a probability beyond σ_j.

The minMS problem can be expressed in the form of integer program.

$$\text{Min } C = \sum_{i=1}^{n} c_i x_i \tag{1}$$

$$\text{ST: } 1 - \prod_{i=1}^{n} x_i(1 - \sigma_{ij}) \geq \sigma_j, \forall j \tag{2}$$

$$x_i \in \{0, 1\}, i = 1...n \tag{3}$$

$$c_i \geq 0, i = 1...n \tag{4}$$

$$0 \leq \sigma_{ij}, \sigma_j \leq 1, i, j = 1...n \tag{5}$$

In the above model, $c_i, \sigma_{ij}, \sigma_j$ are given variables, where c_i is the cost of mine i, σ_{ij} is the failure probability of target j due to the explosion of mine i, and σ_j is the threshold probability of target j. x_i is the indicator variable to be determined. $x_i = 1$ if mine i is selected; $x_i = 0$ otherwise. The purpose of the optimization is to determine the variables x_i to minimize the total cost C subject to the constraints 2-5. Constraint 2 states that the failure probability of target j resulted from the aggregated blasts should exceed σ_j.

Constraint 2 can be transformed by applying the logarithmic operation. let $v_{ij} = -log_2(1-\sigma_{ij}), r_j = -log_2(1-\sigma_j)$, then Constraint 2 becomes

$$\sum_{\{i|x_i=1\}} v_{ij} = \sum_{i=1}^{n} x_i v_{ij} \geq r_j, \forall j \tag{6}$$

Based on this transformation, the following theorem gives the hardness result.

Theorem 1: The problem of minMS is NP-hard.

Proof: The problem of minMS can be proved to be NP-hard via a reduction from the set-cover problem [14], which can be stated as follows. Given a universe $U = \{e_1, ..., e_n\}$ of n elements, a collection of subsets of U, $S = \{s_1, ..., s_k\}$, where the cost of subset s_i is $c_i, i = 1...k$, find a minimum cost subcollection of S that coves all elements of U.

To see the reduction, let $\sigma_{ij} = \frac{1}{2}$ if mine i covers target j, and $\sigma_{ij} = 0$ otherwise. Then $v_{ij} = 1$ if mine i covers target j, and $v_{ij} = 0$ otherwise. Further let $\sigma_j = \frac{1}{2}$, then $r_j = 1$. Then Constraint 6 becomes $\sum_{\{i|e_j \in s_i\}} x_i \geq 1, \forall e_j$. With mine mapped to the set and target mapped to the element, this is equivalent to say that for any element e_j, at least one set that covers e_j is chosen, which is exactly the constraint of the set cover problem. Therefore, the set-cover problem can be reduced to a special case of minMS problem. Since the set cover problem is known NP-hard, the minMS problem is also NP-hard. ■

The transformation not only helps establish the hardness result, but also provides us with a new perspective to view the minMS problem. While the original Constraint 2 involves the multiplicative operation, which is complicated to solve, we transform it to Constraint 6 after applying the logarithm operation, which only needs additive operation. This motivates us to design a *bucket-tub* model to solve the minMS problem based on the transformed constraint.

In the bucket-tub model, the whole network is represented by a bipartite graph, with the mines on the left and the targets on the right. Each mine i is modeled as a set of buckets each of which can supply $v_{ij} = -log_2(1 - \sigma_{ij})$ volume of water to target j, and each target j is modeled as an empty bucket which can hold up to $r_j = -log_2(1-\sigma_j)$ volume of water. The link between mine i and target j can be seen as the channel through which the water could flow from the buckets to the tubs. Then the problem becomes how to find the minimum-cost subset of bucket sets on the mine side to fill up the empty tubs on the target side.

The following notations are defined for the bucket-tub model, and used throughout the paper.

- s_i: bucket set i.
- S: the collection of bucket sets, i.e., $S = \{s_i | i = 1...n\}$.
- t_j: tub j.
- T: the collection of tubs, i.e., $T = \{t_j | j = 1...m\}$.
- v_{ij}: the volume of water in s_i available for tub t_j.
- v_i: the total volume of water in s_i, i.e., $v_i = \sum_{j=1}^{m} v_{ij}$.
- r_j: the residue space of tub t_j.
- c_i: the cost of bucket set s_i.

In the bucket-tub model, $v_{ij} = -log_2(1 - \sigma_{ij}), r_j = -log_2(1 - \sigma_j)$, where σ_{ij} and σ_j are the parameters of the original problem. Without loss of generality, it is assumed that $v_{ij} \leq r_j$ (otherwise, we can reduce v_{ij} to r_j without affecting the result), and $\sum_{i=1}^{n} v_{ij} \geq r_j$ (otherwise, the requirement of the tub t_j can never be satisfied).

Fig. 1(b) shows the bucket-tub graph corresponding to Fig. 1(a). Each mine is modeled by a set of buckets and each target is modeled by an empty tub. Each bucket can provide v_{ia}, v_{ib} volume of water to the tub t_a and t_b, whose residue space is r_a, r_b, respectively. The purpose is to select the minimum-cost bucket sets to fill the tubs t_a, t_b with water. In general, the problem can be reformulated using the bucket-tub model as follows:

Given a bipartite graph $G(S,T)$, where $S = \{s_i | i = 1...n\}, T = \{t_j | j = 1...m\}$, together with the parameters $v_{ij}, r_j, c_i, i = 1...n, j = 1...m$, we want to find the minimum-cost subcollection of bucket sets to fill all the tubs, i.e., $\sum_{i=1}^{n} x_i v_{ij} \geq r_j, \forall j$.

The new problem formulation could help us avoid the complex form of the original formulation and motivates our

design of the algorithms based on the bucket-tub model in the next section.

IV. ALGORITHMS AND THEIR PERFORMANCE BOUND

In this section, we present two approximation algorithms based on the bucket-tub model, i.e., greedy algorithm, layering algorithm, and prove their performance bound. We use S_{gre}, S_{lay} to denote the solution set produced by each algorithm, and use GRE, LAY to denote their corresponding cost, respectively. For example, for the greedy algorithm $GRE = \sum_{s_i \in S_{gre}} c_i$. Here, the subscript takes the form of $s_i \in S_{gre}$ as an abbreviation of $\{i|s_i \in S_{gre}\}$, and the same form of abbreviation will be applied in the remainder of the paper.

A. Greedy Algorithm

Algorithm 1 Greedy Algorithm

Input: a bucket-tub graph $G(S,T)$
Output: a subcollection of bucket sets and its cost, i.e., S_{gre} and GRE
Procedure:
1: $S_{gre} = \emptyset$
2: **while** $T \neq \emptyset$ **do**
3: calculate $e_i = \frac{c_i}{v_i}$ for each $s_i, i = 1...n$.
4: pick the most effective set s_k with $e_k = \text{mine}_i$, i.e., $S_{gre} = S_{gre} + \{s_k\}$.
5: /*update r_j and v_{ij} of the remaining bucket sets and tubs*/
6: **for** $(j \in \{j|v_{kj} > 0\})$
7: $r_j = \max(r_j - v_{kj}, 0)$
8: **for** $(i \in \{i|v_{ij} > 0\})$
9: $v_{ij} = \min(v_{ij}, r_j)$
10: **end for**
11: **end for**
12: remove any tub whose residue space is 0, i.e., $T = T - \{t_j|r_j = 0\}$.
13: remove any bucket set whose connected tubs are all full, i.e., $S = S - \{s_i|v_i = 0\}$.
14: **end while**
15: Output S_{gre} and GRE.

The greedy algorithm is executed in iterations. During each iteration, it picks the most cost-effective bucket set, and fills its water to the connected tubs. We use e_i to denote the *average cost* of s_i, which is defined as $e_i = c_i / \sum_{j=1}^{m} v_{ij} = c_i / v_i$. Then the most cost-effective set is the set with the minimum average cost. Each time a set s_k is chosen, r_j and v_{ij} need to be updated (as in line 6-11 of Algorithm 1). The whole process is then repeated iteration after iteration, until all the tubs become full. The pseudo code is summarized in Algorithm 1 .

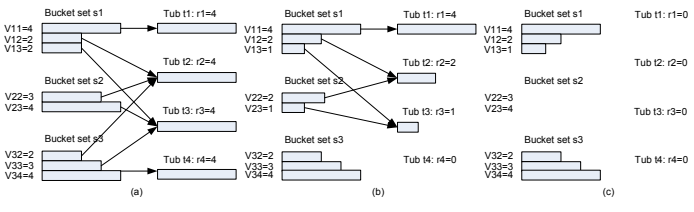


Fig. 3. An example of the greedy algorithm

Fig. 3 uses an example to illustrate the greedy algorithm. There are three bucket sets and four empty tubs. Each bucket set has cost 1, with the volume size and residue space as given in Fig. 3(a). According to Algorithm 1, in the first iteration (Fig. 3b), s_3 is chosen because it is the most effective set, i.e., $e_3 = \frac{1}{4+3+2} = \frac{1}{9}$. Following that, r_i, v_{ij} will be updated accordingly. For example, $r_3 = 4 - 3 = 1, v_{23} = \min(4,1) = 1$. In the second iteration, s_1 is chosen because $e_1 = \frac{1}{7} < e_2 = \frac{1}{3}$. Since s_1, s_3 together can fill the space of all the tubs (Fig. 3c), the algorithm terminates with the total cost of 2.

B. Layering Algorithm

Algorithm 2 Layering Algorithm

Input: a bucket-tub graph $G(S,T)$
Output:a subcollection of bucket set and its cost, i.e., S_{lay} and LAY
Procedure:
1: $k = 0, G(0) = G, S(0) = S, T(0) = T, S_{lay} = \emptyset$
2: $RC_i(0) = c_i, \forall i \in \{i|s_i \in S\}$
3: /*decompose graph into layers*/
4: **while** $T(k) \neq \emptyset$ **do**
5: identify the zero bucket set $Z_b(k) = \{s_i(k)|v_i(k) = 0\}$.
6: $S(k+1) = S(k) - Z_b(k)$.
7: calculate $e_{min}(k) = \text{mine}_i(k) = \min \frac{RC_i(k)}{v_i(k)}$.
8: pick $P(k) = \{s_i|e_i(k) \in [e_{min}(k), \alpha \cdot e_{min}(k)], \alpha \geq 1\}$.
9: $S(k+1) = S(k) - P(k)$.
10: /*update c_i, v_{ij}, r_j of the remaining bucket sets and tubs*/
11: **for** $(i \in \{i|s_i \in S(k)\})$
12: $c_i(k) = \min(c_i(k), \alpha \cdot e_{min}(k) \times v_i(k))$
13: $RC_i(k+1) = RC_i(k) - c_i(k)$
14: **end for**
15: **for** $(j \in \{j|v_{lj}(k) > 0, s_l \in P(k)\})$
16: $r_j(k+1) = \max(r_j(k) - v_{lj}(k), 0)$
17: **for** $(i \in \{i|v_{ij}(k) > 0\})$
18: $v_{ij}(k+1) = \min(v_{ij}(k), r_j(k))$
19: **end for**
20: **end for**
21: identify the zero tub set $Z_t(k) = \{t_j|r_j(k) = 0\}$.
22: $T(k+1) = T(k) - Z_t(k)$.
23: $k = k + 1$.
24: **end while**
25: Output $S_{lay} = \sum_{k=0}^{L-1} P(k)$ and LAY.

As shown in Fig. 4(a), the layering algorithm decomposes the given graph into layers, and each layer is a bucket-tub graph by itself. At each layer, the cost of any set is no greater than $\alpha \cdot e_{min}(k)$, where $\alpha \geq 1$ is the *relaxation factor* and $e_{min}(k)$ is the minimum average residue cost at layer k. Denote the graph of layer k by $G(k)$. Similarly, the same method is used to extend the definition of other variables. For example, the variables $S(k), T(k), c_i(k), v_{ij}(k), v_i(k), r_j(k)$ all preserve the original definition, but specifically refer to layer k.

Algorithm 2 lists the pseudo code of the layering algorithm and Fig. 4(a) shows its illustrative diagram. In the beginning, $G(0) = G, S(0) = S, T(0) = T$. Following that, the layers are constructed one after another. For example, $G(k+1)$ is built upon $G(k)$ as follows. First, remove from $S(k)$ the bucket sets which have zero volume of water, and use $Z_s(k)$ to denote the collection of such sets, i.e., $Z_s(k) = \{s_i|v_i(k) = 0, s_i \in S(k)\}$. Second, define $RC_i(k) = c_i - \sum_{l=0}^{k-1} c_i(l)$ to be the *residue cost* of s_i at layer k, and $e_i(k) = \frac{RC_i(k)}{v_i(k)}$ to be the *average residue cost* of s_i at layer k. Then calculate the minimum average residue cost by $e_{min}(k) = \min_i \frac{RC_i(k)}{v_i(k)}$. Further identify and pick the collection $P(k) = \{s_i|e_i(k) \in [e_{min}(k), \alpha \cdot e_{min}(k)], \alpha \geq 1\}$, which are removed from $S(k)$. Then we have $S(k+1) = S(k) - P(k) - Z_s(k)$. Third, calculate the cost of each bucket set s_i at layer k,

i.e., $c_i(k) = \min(c_i(k), \alpha \cdot e_{min}(k) \times v_i(k))$. It is followed that the average residue cost of each set at layer k is at most $\alpha \cdot e_{min}(k)$. $r_j(k)$ and $v_{ij}(k)$ are updated in a manner similar to the greedy algorithm. Finally, remove from $T(k)$ the collection of tubs with zero residue space, denoted by $Z_t(k)$. Then we have $T(k+1) = T(k) - Z_t(k)$. The entire process will repeat layer after layer until the residue space of all the tubs is filled. Suppose the graph is decomposed into total $L+1$ layers, as shown in Fig. 4(a), then the collection $S_{lay} = \sum_{k=0}^{L-1} P(k)$ will be the output.

From the above description, it can be seen that the cost of each bucket set is decomposed into layers. Suppose a bucket set s_i stays in the bucket-tub graph until layer k, then we have $c_i = \sum_{l=0}^{k} c_i(l)$. At each layer l, we pick the sets whose average residue cost falls below $\alpha \cdot e_{min}(l)$. Therefore, the relaxation factor α determines how many bucket sets can be chosen per layer. When a larger α is used, more bucket sets can be chosen at each layer, so the graph is decomposed into fewer layers and involves less computation overhead. On the other hand, when α increases, the approximation ratio of the algorithm will increase, as shown in subsection IV-C. Therefore, an appropriate relaxation factor is required to strike a balance between the computational burden and performance bound.

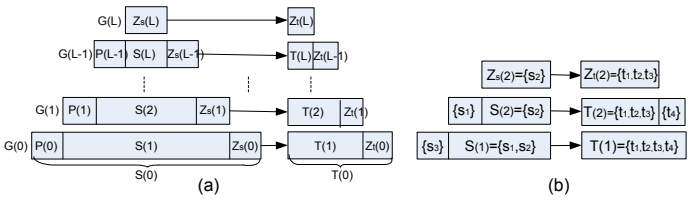


Fig. 4. Illustration of the layering algorithm (a) the general case, where a given graph G is decomposed into $L+1$ layers. (b) a special case, taking Fig. 3 as an example

Fig. 4(b) uses the same example of Fig. 3 to illustrate the layering algorithm, assuming $\alpha = 1$. In the beginning, $G(0) = G$. $P(0) = \{s_3\}$ because $s_3(0)$ has the smallest average residue cost, i.e., $e_{min}(0) = e_3(0) = \frac{1}{9}$. $Z_s(0) = \emptyset$. Then $c_1(0) = e_{min}(0) \times v_1(0) = \frac{8}{9}, c_2(0) = \frac{7}{9}$, and r_j, v_{ij} is updated in the same way as in Fig. 3(b). At layer 1, $P(1) = \{s_1\}$ because $e_{min}(1) = e_1(1) = \frac{(1-\frac{8}{9})}{7} = \frac{1}{63}$. As seen from Fig. 3(b), $r_4(1) = 0$, thus $Z_t(1) = \{t_4\}$. Since $T(3) = \emptyset$, the graph is decomposed into three layers, and the algorithm outputs $S_{lay} = P(0) + P(1) = \{s_1, s_3\}$. On the other hand, if we set $\alpha = 2$, a similar calculation will show that $S_{lay} = \{s_1, s_2, s_3\}$ is the output, with only one layer constructed.

Although the layering algorithm looks more complicated than the greedy algorithm, it has the same worst-case computational complexity $O(n^2+mn)$. In addition, it has a provable approximation ratio that relies only on the relaxation factor and the maximum number of mines connected with a tub, as shown in the next subsection.

C. Performance Bound

In this section, we will establish the performance bound for the layering algorithm and greedy algorithm. We will first prove that the layering algorithm has an approximation ratio of $\alpha \cdot f$, where f is the maximum number of sets/mines that a tub/target is associated with, and α is the tunable relaxation factor. After that, we will show that the greedy algorithm has an approximation ratio of $\sum_j R_j$, where R_j is the coefficient in the related integer program.

Lemma 1: In the layering algorithm, given a graph $G(k)$ at layer k, where the average residue cost of all the bucket sets falls into $[e_{min}(k), \alpha e_{min}(k)], \alpha \geq 1$, there is $\sum_{s_i \in S(k)} c_i(k) \leq \alpha \cdot f \cdot \text{OPT}(G(k))$, where $\text{OPT}(G(k))$ is the cost of the optimal solution to $G(k)$.

Proof:

At layer k, because $e_i(k) \in [e_{min}(k), \alpha e_{min}(k)]$, there is $c_i(k) = e_i(k) \times v_i(k) \leq \alpha \cdot e_{min}(k) v_i(k)$. Further because $v_{ij} \leq r_j$, and each tub is connected with at most f buckets, there is $\sum_{s_i \in S(k)} v_i(k) \leq f \cdot \sum_{t_j \in T(k)} r_j(k)$. In other words, the total volume of water that the buckets can provide is no greater than f times the total space volume of the tubs. Then we have

$$\sum_{s_i \in S(k)} c_i(k) \leq \alpha \cdot f \cdot e_{min}(k) \sum_{t_j \in T(k)} r_j(k) \tag{7}$$

Use $S_{opt}(k)$ to denote the collection of bucket sets chosen in the optimal solution for $G(k)$. It is followed that the total water provided by the buckets in $S_{opt}(k)$ can fill the tubs in $T(k)$, i.e., $\sum_{s_i \in S_{opt}(k)} v_i(k) \geq \sum_{t_j \in T(k)} r_j(k)$. Then we have

$$\text{OPT}(G(k)) = \sum_{s_i \in S_{opt}(k)} c_i(k) \geq e_{min}(k) \times \sum_{t_j \in T(k)} r_j(k) \tag{8}$$

Combining Eqn. 7 & 8 gives the lemma.

■

Lemma 2: Suppose S_0 denotes an arbitrary solution to G, i.e., the bucket sets in S_0 can collaboratively fill the tubs in G. Then $S_0 \cap G(k)$ is a solution to $G(k)$ in the layering algorithm.

Proof: Since S is a solution to G, for any tub t_j in G, $\sum_{s_i \in S_0} v_{ij} \geq r_j$. We divide the solution S_0 in two parts, i.e., $S_0 = (S_0 \cap G(k)) + (S_0 - S_0 \cap G(k))$. In the following, we will prove that the collection $(\sum_{l<k} P(l) + S_0 \cap G(k))$ is a solution to G. Define $Q(j) = \{s_i | s_i \in (S_0 - S_0 \cap G(k)), v_{ij} > 0\}$ for each t_j. We want to show that any tub t_j associated with $Q(j)$ can be filled by the buckets in $(\sum_{l<k} P(l) + S_0 \cap G(k))$.

Given a tub t_j, for any set $s_i \in Q(j)$, there are only two possible cases, i.e., s_i belongs to some removed set ($s_i \in Z_s(l), l < k$) or s_i belongs to some chosen set ($s_i \in P(l), l < k$). If it is the former case, it implies that tub t_j has already been filled till layer l, i.e., $r_j(l) = 0$. Thus without loss of generality, assume all sets $s_i \in Q(j)$ belong to the latter case, i.e., $s_i \in \sum_{l<k} P(l)$. It is followed that $Q(j) \subseteq \sum_{l<k} P(l)$ for each j. Then we have

$$\sum_{s_i \in \sum_{l<k} P(l)} v_{ij} + \sum_{s_i \in S_0 \cap G_k} v_{ij} \geq \sum_{s_i \in (S_0 - S_0 \cap G_k)} v_{ij} + \sum_{s_i \in S_0 \cap G(k)} v_{ij}$$

$$= \sum_{s_i \in S_0} v_{ij} \quad \geq \quad r_j, \text{for each } j \tag{9}$$

Eqn. 9 shows that the set $(\sum_{l<k} P(l) + S_0 \cap G(k))$ is a solution to G. Therefore, $S_0 \cap G(k)$ must be a solution to $G(k)$. ■

Theorem 2: The layering algorithm has an approximation ratio $\alpha \cdot f$, i.e., LAY $\leq \alpha \cdot f \cdot$ OPT, where OPT is the cost of the optimal solution to G.

Proof: Use S_{opt} to denote the collection of bucket sets chosen in the optimal solution to G. Because in the layering algorithm, each set decomposes its cost into different layers, for any set s_i chosen at layer k we have

$$c_i = \sum_{l \leq k} c_i(l) = \sum_{l \leq k} e_{min}(l) v_i(l), \text{ for } \forall s_i \in S(k) \tag{10}$$

Suppose G is decomposed into $L+1$ layers, as shown in Fig. 4(a). Then, the total cost of the sets chosen by the layering algorithm is

$$LAY = \sum_{s_i \in \sum_{k<L} P(k)} c_i = \sum_{s_i \in \sum_{k<L} P(k)} \sum_{l \leq k} e_{min}(l) v_i(l) \tag{11}$$

In Eqn. 11, the total cost is first summed vertically (over different layers), and then horizontally (over different sets) in Fig. 4(a). Alternatively, we can calculate by summing first horizontally (over different sets within a layer) and then vertically (over different layers). Combining the fact that $P(k) = S_{lay} \cap G(k)$, we have

$$LAY = \sum_{k<L} \sum_{s_i \in P(k)} c_i(k) = \sum_{k<L} \sum_{s_i \in (S_{lay} \cap G(k))} c_i(k) \tag{12}$$

Following the similar steps of Eqn. 10- 12, we now calculate the total cost of the optimal solution. For any set $s_i \in Z_s(k)$ removed at layer k, because $c_i(k) \geq e_{min}(k) v_i(k) = 0$, we have

$$c_i = \sum_{l<k} c_i(l) + c_i(k) \geq \sum_{l<k} e_{min}(l) v_i(l), \text{ for } s_i \in Z_s(k) \tag{13}$$

Therefore, the total cost of the optimal solution is

$$OPT = \sum_{k \leq L} \sum_{s_i \in (S_{opt} \cap G(k))} c_i(k) \geq \sum_{k<L} \sum_{s_i \in (S_{opt} \cap G(k))} c_i(k) \tag{14}$$

On the other hand, at layer k for any solution $S_0(k)$ to $G(k)$, we have

$$\sum_{s_i \in S_{opt}} c_i(k) \leq \sum_{s_i \in S_0(k)} c_i(k) \leq \sum_{s_i \in S(k)} c_i(k) \tag{15}$$

According to Lemma 2, both $(S_{lay} \cap G(k))$ and $(S_{opt} \cap G(k))$ are solutions to $G(k)$. Thus, by Eqn. 15 and Lemma. 1, we get

$$\sum_{s_i \in (S_{lay} \cap G(k))} c_i(k) \leq \alpha \cdot f \cdot \sum_{s_i \in (S_{opt} \cap G(k))} c_i(k) \tag{16}$$

Finally, following Eqn. 12, 14 , 16, we have

$$LAY \leq \alpha \cdot f \cdot OPT \tag{17}$$

To establish the performance bound of the greedy algorithm, we can exploit some approximation result on integer programming [15], [16]. However, to do this, the coefficients in Constraint 6, i.e.,v_{ij}, r_j have to be integers. Observe that when v_{ij} and r_j are multiplied by the same constant, the integer programming model in Section III remains unchanged. Thus, we can safely scale the problem to convert the coefficients to integers via some existing techniques such as [17]. Then we get the following theorem.

Theorem 3: The greedy algorithm has an approximation ratio of $\sum_j R_j$, where R_j is the integer coefficient corresponding to r_j after scaling the Constraint 6.

The proof of Theorem 3 directly follows the result of [16] which states that applying the greedy heuristic to any integer program with the form of MIN(cx) subject to $Ax \geq b$ and $x \in \{0,1\}$ has an approximation ratio of $H(k)$, where $k = \sum_j b_j$.

V. DISTRIBUTED IMPLEMENTATION

In this section, we will present a distributed implementation of the greedy algorithm. The aforementioned greedy algorithm iteratively picks up the most cost-effective set and updates the residue space of the tubs and the available water in the buckets. We call it the *global greedy algorithm*, which requires centralized knowledge. Nevertheless, based on a newly defined *bucket graph*, we could design a *local greedy algorithm* which involves only the local message exchange and achieves the same performance as the global version of the algorithm.

In a bucket graph, each node represents a bucket set. Two nodes have a link with each other if the two bucket sets share a common neighboring tub in the corresponding bucket-tub graph. For example, Fig. 5(a) shows the bucket graph corresponding to the smart-mine network in Fig. 1(a). Although a bucket graph should be composed only of the bucket sets and the links between them, in Fig. 5 we explicitly draw the targets (with residue space r_a and r_b) and use dashed lines to depict the relationship between the bucket sets and tubs for a better illustration. The weight over each dashed line, i.e., v_{ij}, denotes the water available in the bucket set for the specific tub. In reality, all these information needs to be cached at each node. The observation in the following lemma can help us design a *local greedy algorithm* based on the bucket graph, which can be proved to produce the same solution set as the global greedy algorithm.

Lemma 3: Given a bucket graph G_b, if a bucket set represented by a node is the most cost-effective set among its neighbors, it will be part of the solution set chosen in the global greedy algorithm.

Proof: Suppose a node s has the minimum average cost among its neighbors, denoted by $N(s)$. Then no node among

$N(s)$ shall be picked before s is picked in the global greedy algorithm. This is because the average cost $e_i = \frac{c_i}{v_i}$ at each node i is a non-decreasing function as other nodes are picked. Therefore, s remains to be the most cost-effective among its neighbors. In addition, to fill the tubs associated with s, at least one node among s and $N(s)$ has to be picked. Thus, eventually s will be chosen by the global greedy algorithm. ■

We assume the communication range of the sensor is at least twice the range of the explosion[1]. Thus, two neighboring nodes in the bucket graph can communicate directly. The local greedy algorithm is executed in iterations. During each iteration, the nodes which have the smallest average cost among their neighbors will elect themselves. Then, information about v_{ij}, r_j is updated in a distributed manner, after which another iteration will begin. From the algorithmic point of view, the nodes elected per iteration constitute an Independent Set, which is defined as a subset of nodes among which there is no link between any two nodes. The set is a maximal independent set (MIS) if no more nodes can be added to generate a bigger independent set [18]. Therefore, our algorithm is equivalent to find a MIS by electing the most effective nodes among their neighbors in each iteration.

Algorithm 3 Distributed Greedy Algorithm

Input: node s, its neighbors $N(s)$, and its connected tub set T_s
Output: election or non-election status
For Each Node s:
1: **while** $T_s \neq \emptyset$ **do**
2: **if** s has the smallest average cost among its neighbors **then**
3: s elects itself, broadcasts $msg(s, ELE, c_s)$, and exits.
4: **end if**
5: **if** $msg(id, ELE, cost)$ is received **then**
6: update the residue space in $T_s \cap T_{id}$, and remove the tubs with zero residue space from T_s.
7: recalculate c_s, and broadcast $msg(s, UPD, c_s)$. Go to 2
8: **end if**
9: **if** $msg(id, UPD, cost)$ is received **then**
10: update the average cost of neighbor s_{id}. Go to 2.
11: **end if**
12: **end while**
13: output the non-election status.

The pseudo code of the local greedy algorithm is listed in Algorithm 3. The message format is defined as $msg(id, type, cost)$, where id and $cost$ are the fields related to the sender, and $type$ can be set to be ELE or UPD, which corresponds to the election notice or the update report. An election notice is sent out when a node finds itself to be the most cost-effective set among its neighbors (line 2-4). When an election notice is received, the average cost of the receiver is updated in a manner similar to the global greedy algorithm, after which an update report is sent out (line 5-8). When an update report is received, the average cost of the sender is simply updated to the *cost* value contained in the message (line 9-11).

Theorem 4: The local greedy algorithm is guaranteed to terminate, and upon termination, the same solution set is produced as in the global version of the algorithm.

[1]The normal communication range of a sensor is around tens of meters; the explosion range is less than ten meters for certain types of mine.

Proof: At any instant, there is at least one node which has the smallest average cost among its neighbors and thus is qualified to elect itself. Therefore, the algorithm will not stop until the requirements of all the tubs are satisfied. In addition, Lemma 3 assures that the same solution set will be produced as in the global greedy algorithm upon termination, although the bucket sets may be picked in a different order. ■

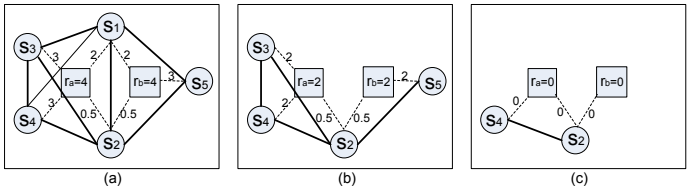


Fig. 5. An example to illustrate the local greedy algorithm. (a) the bucket graph corresponding to Fig. 1. (b) the bucket graph after node s_1 is elected in the first iteration. (c) the bucket graph after nodes s_3, s_5 are elected in the second iteration.

Fig. 5 uses a small example to illustrate the local greedy algorithm. In Fig. 5(a), there are five bucket sets (circle node), which are connected with two targets (square node). Each bucket set has cost 1 and can provide v_{ij} volume of water for each tub, as shown over the dashed arrows. Each tub has the empty space of 4. In the first iteration, s_1 elects itself since it has the smallest average cost among its neighbors (tie is broken by id). Then v_{ij}, r_a, r_b are updated and the algorithm continues to run among the remaining nodes (Fig. 5b). Similarly, in the second iteration, s_3, s_5 elect themselves, after which the tubs become full (Fig. 5c).

Like the layering algorithm, the election criteria in the local greedy algorithm can also be relaxed to reduce the number of iterations required. We call it the *fast local greedy algorithm*. Instead of electing only the nodes with the smallest average cost among its neighbors, any node whose average cost falls in the range of $[e_{min}, \alpha e_{min}]$ can be elected, where e_{min} is the minimum average cost among the neighbors and $\alpha \geq 1$ is the relaxation factor. To implement this, we need to change the message format to $msg(id, type, cost, e_{min})$ by adding a field on the minimum average cost. Any node whose average cost is the local minimum will set up the field e_{min} accordingly. For other nodes that receive the election notice with e_{min} properly set, they will check whether their average cost belongs to $[e_{min}, \alpha e_{min}]$ and decide whether to elect themselves. This way, more nodes will be elected per iteration and less total iterations will be needed.

Algorithm 3 is a distributed implementation of the global greedy algorithm. According to Theorem 4, the local greedy algorithm picks the same collection of bucket sets as in the global greedy algorithm, without degrading the performance. On the other hand, the fast greedy algorithm uses the relaxation factor to trade off the performance for the lower message overhead and computational burden. Both the algorithms have the worst case message complexity $O(d)$ per node, where d is the maximum number of neighbors among all the nodes in the bucket graph.

VI. PERFORMANCE EVALUATION

In this section, we verify the effectiveness of the proposed algorithms through simulations. In the simulation, n mines and m targets are randomly deployed in a 10×10 square area, with n varying from 100 to 1000, and m varying from 10 to 100. The blast range and the sensing range are normalized to be 1. The communication range is set to be 2, which is twice of the blast range and sensing range. The cost of each mine is uniformly distributed between 1 and 4 units. To emulate the destructive impact on the target, the blast model in Fig. 2 is used. The threshold probability for each target is unified to be 0.9.

Besides the greedy algorithm and layering algorithm, two other simple heuristics are evaluated, i.e., random algorithm and max-weight first algorithm. The random algorithm arbitrarily picks the bucket set, while the max-weight first algorithm picks the bucket set in which a single bucket has the largest volume of water available for the connected tub. This is equivalent to select the mine which has the shortest distance to the target. The experiments are done over a customized C++ simulator and each point is a statistical average of 50 experiments.

A. Comparison of different algorithms

Figs. 6, 7 compare the cost of different algorithms with varying number of mines, when the number of targets is 50 and 100, respectively. It is shown that the greedy algorithm consistently performs best. The layering algorithm stays close to the greedy algorithm when the relaxation factor R is 1. As the relaxation factor increases from 1 to 2, the performance of the layering algorithm becomes worse, but still better than the max-weight first algorithm. Regarding the performance of the random algorithm, a big gap can be observed compared with the other algorithms.

Another observation is that increasing the number of mines may affect the trend of different algorithms in different ways. As the number of mines increases, the cost of the random algorithm tends to increase, due to its pure randomness. However, increasing the number of mines beyond 200 (which is sufficient to destroy all the targets) can cause the other algorithms to gradually decrease. This is because all other algorithms exploit certain level of intelligence when selecting the mines. Thus the more candidate mines available, the more likely a better solution can be produced.

Figs. 8, 9 further compare the cost of different algorithms with varying number of targets, when the number of mines is 500 and 1000, respectively. Regarding the relative performance among different algorithms, the same conclusion can be drawn as in Figs. 6, 7, with the random algorithm to be the worst and the greedy algorithm to be the best. It can be noticed that the cost of all the algorithms increases as the number of targets increases. However, the increase for the random algorithm is roughly linear, but for the other algorithms it is sublinear. This implies that a smarter algorithm such as the greedy algorithm is less affected by the increasing number of targets, because it takes into account both the effect of single explosion on multiple targets and collaborative explosions on a single target.

B. Effect of the relaxation factor

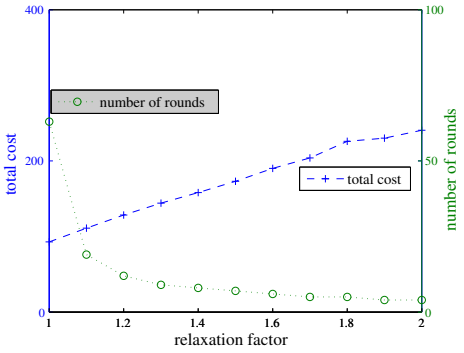


Fig. 12. effect of the relaxation factor R in the fast greedy algorithm($n = 1000, m = 100$)

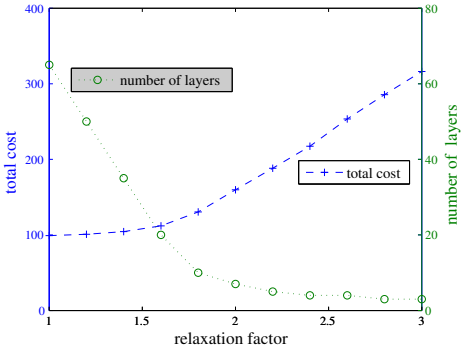


Fig. 13. effect of the relaxation factor R in the layering algorithm ($n = 1000, m = 100$)

Relaxation factor R is an important parameter for the fast greedy and layering algorithm. Figs. 10, 11, 12, 13 use double y-axes to study the effect of relaxation factor R, with the left y-axis to denote total cost and the right y-axis to denote number of rounds/layers. In particular, Fig. 10 shows the effect of R in the fast greedy algorithm, when $m = 50, n = 500$. As R increases, the cost of the fast greedy algorithm increases, but the number of its execution rounds reduces. For example, when R increases from 1.0 to 1.1, its cost rises from 73 to 84 while its computation rounds reduce from 41 to 16. This is expected since the value of the relaxation factor R reflects the strictness of the selection criteria. When the criteria is relaxed, i.e., with larger R, more mines can be picked per round thus less number of total rounds are required.

Fig. 11 shows the effect of R in the layering algorithm, when $m = 50, n = 500$. Similar to Fig. 10, as R increases, the cost of the layering algorithm increases, but the number of layers reduces.

Figs. 12, 13, where $m = 100, n = 1000$, lead to the same conclusion as Figs. 10, 11. Since in both algorithms, the message overhead or the computational burden is proportional to the number of rounds/layers, R needs to be appropriately set to strike a balance between different performance metrics.

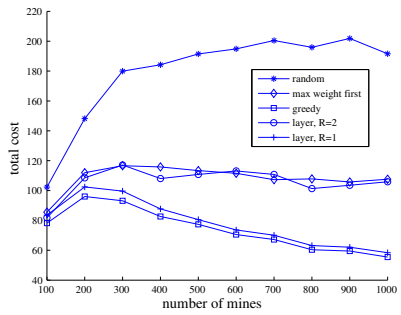


Fig. 6. effect of number of mines n ($m = 50$)

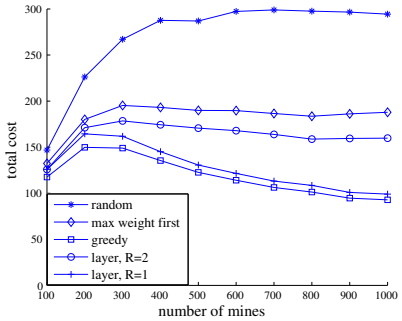


Fig. 7. effect of number of mines n ($m = 100$)

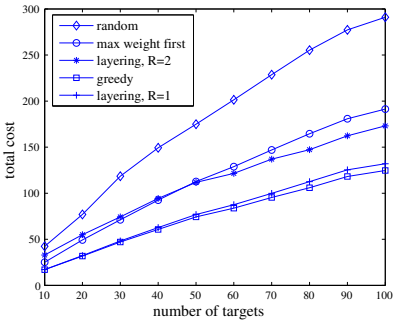


Fig. 8. effect of number of targets m ($n = 500$)

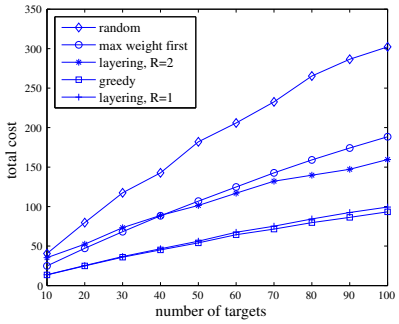


Fig. 9. effect of the number of targets m ($n = 1000$)

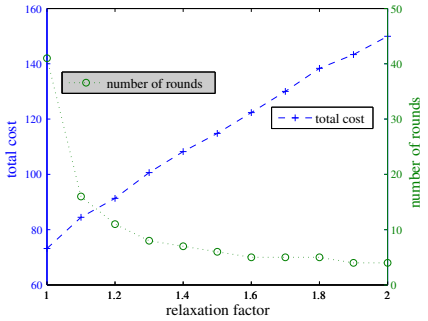


Fig. 10. effect of the relaxation factor R in the fast greedy algorithm ($n = 500, m = 50$)

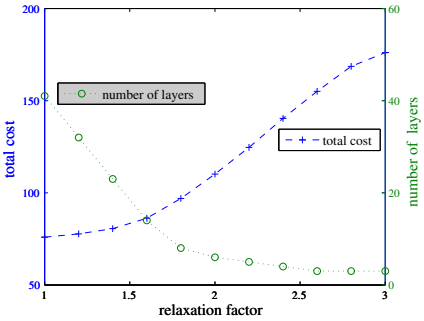


Fig. 11. effect of the relaxation factor R in the layering algorithm ($n = 500, m = 50$)

VII. Conclusions

This paper explores the potential of networking the landmines via sensor technology. Specifically, we formulate a detonable mine selection problem, with the objective to destroy the intruding target using the minimum cost. Due to its NP complexity, we focus on the design of approximation solutions based on a novel bucket-tub model, where mine is mapped to bucket set and target is mapped to tub. Two classes of approximation algorithms are proposed, whose performance bound away from the optimal result is given. Among them, it is shown that the layering algorithm can achieve an approximation ratio that relies only on the maximum number of mines/buckets that a target/tub is associated with, and that the greedy algorithm can be implemented in a pure distributed manner without degrading the performance. Theoretical analysis and extensive simulations verified the effectiveness of the proposed algorithms.

Our work is among the first efforts towards filling the gap between the mine industry and impact engineering. More interesting dimensions such as trying different blast models, dealing with the mobile case by taking advantage of the target's mobility pattern can be investigated. As the landmine may linger in the field for a long time, self monitoring and postwar disposal issues are also worth further study.

References

[1] W. Merrill, L. Girod, B. Schiffer, D. McIntire, G. Rava, K. Sohrabi, F. Newberg, J. Elson, and W. Kaiser, "Dynamic networking and smart sensing enable next-generation landmines," *IEEE Pervasive Computing*, 2004.

[2] I. Akyildiz, W. Su, Y. Sankarasubramaniam, and E.Cayirci, "Wireless Sensor Networks: A Survey," *Computer Networks*, 2002.

[3] A. Neubergera, S. Pelesc, and D. Rittela, "Scaling the response of circular plates subjected to large and close-range spherical explosions. part ii: Buried charges," *International Journal of Impact Engineering*, 2007.

[4] N. Jacoba, G. Nuricka, and G. Langdon, "The effect of stand-off distance on the failure of fully clamped circular mild steel plates subjected to blast loads," *International Journal of Impact Engineering*, 2007.

[5] N. Edmondson, "Fatality probabilities for people in the open when exposed to blast," SRD, Tech. Rep. RANN/2/49/00082/90, March 1992.

[6] "Controlling risks around explosive stores: review of the requirements on separation distances," MBTB Limited, Health and Safety Executive, United Kingdom, Tech. Rep., 2002.

[7] P. Volgyesi, G. Balogh, A. Nadas, C. Nash, and A. Ledeczi, "Shooter localization and weapon classification with soldier-wearable networked sensors," in *MobiSys*, 2007.

[8] X. Wang, G. Xing, Y. Zhang, C. Lu, R. Pless, and C. Gill, "Integrated coverage and connectivity configuration in wireless sensor networks," in *ACM SenSys*, 2003.

[9] T. Yan, T. He, and J. A. Stankovic, "Differentiated surveillance for sensor networks," in *ACM SenSys*, 2003.

[10] X. Bai, S. Kumar, D. Xuan, Z. Yun, and T. H. Lai, "Deploying wireless sensors to achieve both coverage and connectivity," in *MOBIHOC*, 2006.

[11] L. Su, C. Liu, H. Song, and G. Cao, "Routing in intermittently connected sensor networks," in *ICNP*, 2008.

[12] M. Hefeeda and H. Ahmadi, "A probabilistic coverage protocol for wireless sensor networks," in *ICNP*, 2007.

[13] M. Li and Y. Liu, "Rendered path: range-free localization in anisotropic sensor networks with holes," in *MobiCom*, 2007.

[14] M. H. Alsuwaiyel, *Algorithms Design Techniques and Analysis*. World Scientific Publishing Company, 1999.

[15] S. Rajagopalan and V. V. Vazirani, "Primal-dual rnc approximation algorithms for set cover and covering integer programs," *SIAM J. Comput.*, 1999.

[16] G. Dobson, "Worst-case analysis of greedy heuristics for integer programming with nonnegative data," *Mathematics of Operations Research*, 1982.

[17] T. J. Cole, "Scaling and rounding regression coefficients to integers," *Applied Statistics*, 1993.

[18] S. Basagni, "A distributed algorithm for finding a maximal weighted independent set in wireless networks," in *PDCS*, 1999.