# A Knapsack Approach to Sensor-Mission Assignment with Uncertain Demands

Diego Pizzocaro[a], Matthew P. Johnson[b], Hosam Rowaihy[c],
Stuart Chalmers[d], Alun Preece[a], Amotz Bar-Noy[b], Thomas La Porta[c]

[a]School of Computer Science, Cardiff University, UK;
[b]Department of Computer Science, City University of New York, US;
[c]Department of Computer Science and Engineering, Pennsylvania State University, US;
[d]Department of Computing Science, University of Aberdeen, UK

## ABSTRACT

A sensor network in the field is usually required to support multiple sensing tasks or missions to be accomplished simultaneously. Since missions might compete for the exclusive usage of the same sensing resource we need to assign individual sensors to missions. Missions are usually characterized by an uncertain demand for sensing resource capabilities. In this paper we model this assignment problem by introducing the Sensor Utility Maximization (SUM) model, where each sensor-mission pair is associated with a utility offer. Moreover each mission is associated with a priority and with an uncertain utility demand. We also define the benefit or profit that a sensor can bring to a mission as the fraction of mission's demand that the sensor is able to satisfy, scaled by the priority of the mission. The goal is to find a sensor assignment that maximizes the total profit, while ensuring that the total utility cumulated by each mission does not exceed its uncertain demand. SUM is NP-Complete and is a special case of the well known Generalized Assignment Problem (GAP), which groups many knapsack-style problems. We compare four algorithms: two previous algorithms for problems related to SUM, an improved implementation of a state-of-the-art pre-existing approximation algorithm for GAP, and a new greedy algorithm. Simulation results show that our greedy algorithm appears to offer the best trade-off between quality of solution and computation cost.

**Keywords:** sensor mission assignment, knapsack problem, uncertain demands, sensor allocation, generalized assignment problem, utility maximization, greedy algorithm, sensor networks.

## 1. INTRODUCTION AND MOTIVATIONS

The sensor-mission assignment problem is a hard problem that occurs when we have a sensor network and multiple competing missions that will make use of those sensors. A sensor network consists of a large number of sensing devices that are able to gather certain facets of information regarding their surroundings (e.g. sound, motion, video, etc.). Usually this network of sensors is already deployed in the field and is used to satisfy the information requirements of multiple sensing tasks or missions. Since missions might compete for the exclusive usage of the same sensing resource we need to assign individual sensors to missions. Consider the example in Figure 1 where two missions require to identify two different targets ("Target 1" and "Target 2") that are located in nearby regions on the map: these two missions are competing for the exclusive control of a particular video sensor which could identify either target. The mission to which the video sensor will be assigned will decide to point the camera in a direction that is completely opposite to where the other mission would require it, therefore the sensor can be assigned to only one mission. In this paper we focus on the cases in which sensors cannot be shared by multiple mission, and we assume an homogeneous sensor network (e.g. only video sensors all with the same sensing capabilities).

In each scenario a single sensor will have a different degree of *utility* for a certain mission, since it will offer to it different quality/quantity of information depending on some factors (such as distance from a certain point where the task related to the mission is located). Missions may have different degrees of importance (*priority*) and different levels of required utility from the sensor network (*demand*).

Correspondence email address: D.Pizzocaro@cs.cf.ac.uk

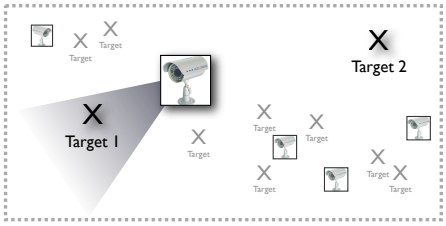


Figure 1: An example of missions competing for the same sensing resource.

Missions are usually characterized by an uncertain demand for sensing resource capabilities because of the unknown conditions on the battlefield (e.g. weather, damage to sensor network, etc.). Consider as an example a mission that requires video sensors to identify a target. If the weather conditions and visibility range in the field are not exactly known (which is often the case), then the required number of sensors and the resolution of their cameras cannot be precisely determined. For instance we might be able to specify only the highest resolution of the cameras needed by the mission or the maximum number of video sensors required: that is what we call an uncertain demand. It is therefore reasonable to require for each mission that the sum of the utility received from the assigned sensors does not exceed its own maximum utility demand. This assumption is based on the principle that sensing resources are in high demand and should not be wasted.[1] In practice a mission could declare a need for sensing capabilities which is much higher than the real information requirement, and for this reason we want to avoid the waste of sensing resources by assigning too many sensors to a mission.

We can model this as a knapsack problem[2] because for each mission we consider its demand as an upper bound not to be exceeded while assigning sensors to it. We can represent a mission with a knapsack associated with a certain capacity that in our case is the maximum utility demand of the mission; the set of sensors can be associated with the set of items that we want to "pack" inside the knapsack, each having a certain "size" that in this case is the utility offer of a sensor to a mission.

Given these assumptions the issue is to choose what is meant by "best assignment of sensors to missions". Since a sensor will contribute different utility to each mission, we can state that each sensor will bring a different benefit to each mission, depending on: the fraction of the mission's demand that the sensor will satisfy, and on the importance of the mission itself. Since the benefit that each sensor can bring to a mission is strongly correlated to the utility that the sensor has for that mission, in our model we will try to maximize the total benefit that an assignment of sensors can bring to all the missions by encouraging the utility to go where it is most useful.

The rest of the paper is organized as follows. Section 2 defines the Sensor Utility Maximization problem which formally models what we have just described. We study its computational complexity showing that it is NP-Complete and a special case of the well known Generalized Assignment Problem (GAP), which groups many knapsack-style problems. In Section 3 we describe four algorithms to solve SUM: two algorithms previously developed for problems related to SUM which we slightly modified for SUM, an improved implementation of a state-of-the-art pre-existing approximation algorithm for GAP, and a new greedy algorithm. In Section 4, we compare the performances of the different algorithms in terms of optimality of the assignment and of the running time required. Section 5 discusses some related work in sensor assignment problems. Finally, Section 6 concludes the paper.

## 2. SENSOR UTILITY MAXIMIZATION MODEL

We model this assignment problem by introducing the Sensor Utility Maximization (SUM) model which can be represented as a complete weighted bipartite graph as shown in Figure 2, in which the vertex sets are composed by sensors $S = \{S_1, \ldots, S_n\}$ and by missions $M = \{M_1, \ldots, M_m\}$. Each mission $M_j$ is associated with a positive-valued priority $p_j$ and with a utility demand $d_j$, which represents the maximum utility demand that a mission might require. Each edge $(S_i, M_j)$ is associated with two values $(e_{ij},\ p_{ij})$. The parameter $e_{ij}$ indicates the utility that sensor $S_i$ could contribute to mission $M_j$. The utility is a very general parameter and may represent different measurements of usefulness: one of these is for example the quality of information (QoI) that a sensor

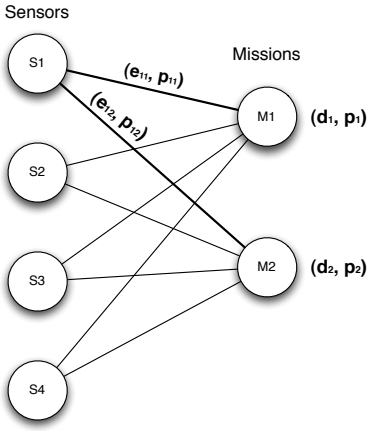


Figure 2: Sensor Utility Maximization model as a bipartite graph.

can provide to a mission, which is usually related to the distance of the sensor from the location of the mission.[3] The value $p_{ij}$ represents the benefit (or profit) that $S_i$ could bring to mission $M_j$. It is equal to the fraction of demand that the sensor will satisfy, scaled by the priority of the mission: $p_{ij} = e_{ij}/d_j \times p_j$. The scaling of $p_{ij}$ by the mission importance allows us to favor high priority missions during the assignment: for example if we have to assign a sensor which can contribute with the same fraction of utility ($e_{ij}/d_j$) to two different missions, then we will assign it to the mission with the highest priority $p_j$.

Our goal is to find an assignment of sensors to missions that maximizes the total benefit (or profit) of the sensor network by assigning the utility where it will be most helpful, while ensuring that the total utility cumulated by each mission does not exceed its maximum utility demand $d_j$. We formally define the problem as:

**Instance:** Given a bipartite graph $G = (S, M, MP, E, SP)$,
where $S = \{S_1, \ldots, S_n\}$ is the vertex set of sensors, $M = \{M_1, \ldots, M_m\}$ is the vertex set of missions, $MP = \{p_1, \ldots, p_m\}$ is the collection of mission priorities, $E = \{(e_{ij}) : \forall$ edges $(S_i, M_j)\}$ is the collection of sensor-mission utility associated with the edges $S \times M$. Finally, $SP = \{(p_{ij}) : \forall\ (S_i, M_j)\}$ is the collection of sensor-mission profits associated with the edges, where $p_{ij} = e_{ij}/d_j \times p_j$.

**Goal:** Find a semi-matching $F \subseteq S \times M$ (i.e. no two chosen edges share the same sensor), in which $\sum_{(S_i,M_j)\in F} p_{ij}$ is maximized for the entire sets S of sensors and M of missions, and where $\sum_{(S_i,M_j)\in F} e_{ij} \leq d_j$ for each mission $M_j$.

It is probably easier to understand the problem in its Integer Linear Programming (ILP) formulation. In the formulation we use one decision variable called $x_{ij}$, indicating if sensor $S_i$ is assigned to mission $M_j$ ($x_{ij} = 1$) or unassigned ($x_{ij} = 0$).

**Maximize:** $\sum_{j=1}^{m} \sum_{i=1}^{n} p_{ij} x_{ij}$ , where $p_{ij} = e_{ij}/d_j \times p_j$.

**Such that:** $\sum_{i=1}^{n} x_{ij} e_{ij} \leq d_j$, for each mission $M_j \in M$,
$\sum_{j=1}^{m} x_{ij} \leq 1$, for each sensor $S_i$, and
$x_{ij} \in \{0, 1\}$, for each variable $x_{ij}$

For each mission $M_j$ we require that the sum of the utility received by $M_j$ does not exceed its own maximum utility demand $d_j$. This assumption is based on the same principle we cited in Section 1 that states that sensing resources are in high demand and should not be wasted.[1]

This constraint highlights also that the SUM model is a knapsack-style problem, and indeed very similar to the well known Multiple Knapsack Problem (MKP).[4] The Knapsack Problem is the maximization problem of

choosing some items that can fit into one bag (of maximum size) to be carried on a trip. We are given a set of items, each with a size and a value, and a knapsack with a given capacity, then we have to determine which items to insert in the knapsack so that: the total size of the chosen items is less than or equal to the knapsack's capacity, and the total value of the chosen items is as large as possible*. The MKP is a similar maximization problem where instead of a single knapsack we have many, each with a different capacity. The objective is to maximize the total value of the items packed into the set of knapsacks, while not exceeding the capacity of each knapsack.

In SUM each mission is a knapsack whose capacity is the max utility demand of the mission ($d_j$), and each sensor is an item whose value is $p_{ij}$ and whose size is $e_{ij}$. SUM differs from MKP because it assumes that each item has a different size and a different value for each knapsack; moreover in SUM size and value are strongly correlated, indeed the value of an item is a function of its size ($p_{ij} = e_{ij}/d_j \times p_j$), which is not the case in MKP. Like MKP, SUM is NP-Complete because it is a generalization of the single knapsack problem which is NP-Complete even when items have strongly correlated sizes and values.[5]

Note that SUM can be also seen as a special case of the Generalized Assignment Problem.[6] This observation will allow us in Section 3 to use an algorithm developed for GAP to solve the SUM problem. The Generalized Assignment Problem assumes that given a set of bins with capacity constraints and a set of items that have a possibly different size and value for each bin, we want to pack a maximum-valued subset of items into the bins. In the GAP general formulation the sizes and values of the items are completely uncorrelated, instead in SUM they are strongly linked as we already noted. Therefore SUM is a special case of GAP where size and value of each item are correlated.

## 3. ALGORITHMS

In this paper we focus on centralized approaches to solve this sensor-mission assignment problem. Distributed solutions are possible but not described here. In a centralized approach all the decisions are taken in a base station, a special control node, therefore we need to collect all the information about the network status in it. This usually requires to exchange a higher number of messages to configure the network in comparison with a distributed approach.[3] The benefit of using a centralized approach is that we have a global overview of the network and because of this we can (usually) find a better solution.

We now describe several methods we developed to solve the SUM problem: we improved the empirical performance of a pre-existent $(2+\epsilon)$-approximation algorithm developed for GAP[6] by using some coding tricks (Section 3.1), we adapted two pre-existent greedy algorithms designed to solve a problem related to SUM[7] (Section 3.2), and finally we developed a new greedy algorithm (Section 3.3).

### 3.1 Pre-existent $(2+\epsilon)$-approximation algorithm for GAP

In Section 2 we showed that SUM is a special case of GAP which is well studied in the literature and for which there exist many approximation algorithms. Here we describe how we combined the approximation algorithm in Cohen et al[6] with the algorithm in Ibarra & Kim[2] obtaining a $(2+\epsilon)$-approximation algorithm.

An algorithm is an $\alpha$-approximation algorithm if $|OPT| \leq \alpha \cdot |ALG|$ for $\alpha \geq 1$, where $|ALG|$ is the value of the objective function for the feasible solution returned by the approximation algorithm, and $|OPT|$ is the value of the objective function for the optimal solution†.

#### ($\alpha$+1)-Approximation Algorithm for GAP

Given any other $\alpha$-approximation algorithm for the knapsack problem, Cohen et al's algorithm (Algorithm 1, above) is an ($\alpha$+1)-approximation algorithm for the Generalized Assignment Problem. The main idea of the

*Here we are actually considering a particular type of knapsack problem called 0-1 knapsack problem, since the items can be chosen ($x_i = 1$) or not chosen ($x_i = 0$). Instead a more general knapsack problem can also decide to take more than only one instance of the same object to insert inside the knapsack.

†It is probably easier to understand the definition of $\alpha$-approximation algorithm if we consider $\alpha \leq 1$, then we have that the condition is: $\alpha \cdot |OPT| \leq |ALG| \leq |OPT|$.

**Algorithm 1** Cohen et al's ($\alpha$+1)-Approximation Algorithm for GAP

**for** each sensor $S_i$ **do**
  $T[i] = -1$
**end for**
**for** each mission $M_j$ **do**
  Call $\alpha$-*Approximation Alg. for Knapsack* to find a solution to mission $M_j$ using the residual profit function $P_j^{RES}$
  **for** each $S_i$ scheduled by the $\alpha$-*Approximation Alg.* to $M_j$ **do**
    $T[i] = j$
  **end for**
**end for**

**Algorithm 2** Ibarra & Kim's $\alpha$-Approximation Algorithm for Knaspack (with $\alpha = \epsilon + 1$)

**for** $v = 1, \ldots, p'_1$ **do**
  $A[1, v] := w_1$
**end for**
**for** $v = p'_1 + 1, \ldots, U$ **do**
  $A[1, v] := +\infty$
**end for**
**for** $l = 2, \ldots, n$ **do**
  **for** $v = 1, \ldots, p'_l$ **do**
    $A[l, v] := \min\{A[l-1, v], w_l\}$
  **end for**
  **for** $v = p'_l + 1, \ldots, U$ **do**
    $A[l, v] := \min\{A[l-1, v], A[l-1, v-p'_l] + w_l\}$
  **end for**
**end for**

algorithm is to iteratively solve the knapsack problem, each time selecting only one of the bins of the GAP model as a knapsack, and considering as items to insert into the knapsack all the items given as input to the GAP. During each iteration, the knapsack problem will use as size for each item the size that each of them has for the current bin, and as profit the residual profit. The residual profit $p_i^{RES}$ of an item $i$ for bin $j$ is defined as: $p_i^{RES} = p_{ij}$ if $i$ is not selected for any other bin, otherwise $p_i^{RES} = p_{ij} - p_{ik}$ if $i$ is selected for bin $k$.

This algorithm has complexity $O(m \cdot f(n) + m \cdot n)$, where $n$ is the number of items (sensors), $m$ is the number of bins (missions) and $f(n)$ is the running time of the $\alpha$-approximation algorithm for the knapsack problem.

### $\alpha$-Approximation Algorithm for Knapsack Problem

Ibarra & Kim's algorithm (Algorithm 2, above) is an efficient (1+$\epsilon$)-approximation algorithm for an arbitrary $\epsilon > 0$, so if we set $\alpha = (1 + \epsilon)$ it can be considered an $\alpha$-approximation algorithm. This algorithm is an FPTAS (Fully Polynomial Time Approximation Scheme) i.e. its time and space complexity grows polynomially with $n$ (number of items) and exponentially with $1/\epsilon$ (both space and time complexity are $O\left(n/\epsilon^2\right)$).

This is a dynamic programming algorithm which defines the elements of a matrix A with $n$ rows (one for each item) and $|ALG|$ columns (one for each possible value of the objective function). Since the value of $|ALG|$ is unknown, the matrix A is defined on $U$ column where $U$ is any upper bound on the value of the optimal solution $|OPT|$. In particular, for $l = 1, \ldots, n$ and $v = 1, \ldots, U$ the element $A[l, v]$ represents the minimum capacity required to obtain a value for the objective function that at least equals $v$ using only the first $l$ items. If it is impossible to get $v$ for the objective function using only the first $l$ items then the rule is to set $A[l, v] = +\infty$. Formally:

$$A[l, v] = \min\{q : \sum_{i=1}^{l} p_i x_i \geq v, \sum_{i=1}^{l} w_i x_i \leq 1, x_i \in \{0, 1\}, i = 1, \ldots, l\}$$

Once all the elements of the matrix $A$ are defined, the value of the optimal solution is:

$$|ALG| := \max\{v : A[n, v] \leq c\}$$

The Ibarra & Kim's algorithm is shown in Algorithm 2 for completeness

Since this algorithm requires that each profit value is an integer, and since the time complexity depends strongly on the maximum item profit, we need to use scaled integer profits different from original item profits. Formally stated, we will use $p'_i = \lfloor p_i/\delta \rfloor$ for $i = 1, \ldots, n$ , where $\delta$ depends on the required approximation error $\epsilon$ in the returned solution ($\delta := \frac{\epsilon \tilde{p}}{n}$, where $\tilde{p} = \max_i\{p_i\}$).

**Algorithm 3** Mission-side Greedy

```
sort the missions in order of decreasing p_j
for each mission M_j in sorted order do
    sort unused sensors in decreasing order of e_ij
    for each unused sensor S_i in sorted order do
        if e_ij + total utility cumulated by M_j ≤ d_j then
            assign S_i to M_j
        end if
    end for
end for
```

**Algorithm 4** Sensor-side Greedy

```
M ← {M_1, ..., M_m}
for each sensor S_i in arbitrary order do
    M_k ← maxarg_{M_j ∈ M}{p_ij}
    (where p_ij = e_ij/d_j × p_j)
    if e_ik + total utility cumulated by M_k ≤ d_k then
        assign S_i to M_k
    end if
end for
```

Finally, to find the real value of the objective function we will have to backtrack on which items are chosen to be inserted into the knapsack and then use the non-scaled profits to compute the original value of the objective function. Therefore, as in all the dynamic programming algorithms, we also need to have another matrix of pointers that will let us backtrack the solution, i.e. to understand which items were inserted in the knapsack. This matrix of pointers will have the same dimensions as the matrix $A$ used to compute the optimal value of the objective function.

#### Combining GAP and Knapsack algorithms

Although the FPTAS has a good asymptotic performance guarantee, when the input values are scaled to a not-unreasonable precision (e.g. with $\epsilon = 0.005$) and a naive implementation is used, the running time can actually become significant. This is especially so in our setting, which involves solving a knapsack problem for each mission. Various coding tricks and improvements have been discovered (and rediscovered) for the basic knapsack algorithms since they appeared in the 1970s. In Appendix A, we describe in detail the specific implementation improvements we used, which allowed our implementation to run in reasonable time on SUM instances of the size we will consider in our experiments (e.g. 1000 sensors and 150 missions).

Combining the GAP $(\alpha + 1)$-approximation algorithm to solve the SUM problem with the Knapsack $\alpha$-approximation algorithm where $\alpha = 1+\epsilon$, leads to a $(2+\epsilon)$-approximation algorithm for the SUM problem where $\epsilon > 0$ is the maximum error in the returned solution (which in our case will be $\epsilon = 0.005$). As stated previously the time complexity of this algorithm is a function of the time complexity of the Knapsack $\alpha$-approximation algorithm used (that is $O\left(\frac{n}{\epsilon^2}\right)$), and in particular it will be $O\left(\frac{mn}{\epsilon^2}\right)$ where $m$ is the number of missions and $n$ is the number of sensors.

### 3.2 Pre-existent Greedy Algorithms adapted for SUM

These two algorithms were inspired by those developed to solve the Semi-Matching with Demands (SMD) problem[3, 7] (see also Section 5) which is a problem closely related to SUM. The general behavior of these greedy algorithms recalls other classic algorithms developed to solve knapsack-style problems. For these algorithms we do not provide theoretical proofs of the degree of approximation of the solution(s) found, we instead measure their empirical performance in Section 4.

#### Mission-side greedy

The first greedy algorithm that we consider (Algorithm 3) takes missions one by one and assigns sensors to them. The approach is to sort missions in order of decreasing priority $p_j$, then for each mission sort the available sensors in decreasing order of utility offered to that mission and assign them until the total utility accumulated by the mission stays under the demand $d_j$. The complexity of this algorithm is $O(mn \log n)$, with $n$ sensors and $m$ missions. For this to be true $m$ must be $O(n \log n)$.

**Sensor-side Greedy**

The second algorithm (Algorithm 4) is a greedy algorithm from the point of view of the sensor. The algorithm assigns each sensor to a mission $M_k$ where that sensor is of most use, i.e. to the mission that maximizes $p_{ij}$. The complexity of this algorithm is $O(nm \log m)$, where "$m \log m$" is given by the fact that we have to find the mission $M_k$ which maximizes $p_{ij}$.

### 3.3 Novel Greedy Algorithm

This algorithm (Algorithm 5) is an improvement of Algorithm 4. The difference is that before processing sensors we first sort them in decreasing order of the maximum profit offer of each sensor, i.e. $max\{p_{ij}\}$ between all the missions $M_j$ to which the sensor can contribute. Note that by having steps 3 and 4 separated in Algorithm 4, we will run into situations where a sensor is not used because the algorithm will only try to assign it once. In this algorithm instead steps 3 and 4 are combined, indeed $M_k$ is assigned the best mission where the cumulated utility does not exceed its demand.

The complexity is still $O(nm \log m)$ if $n$ is $O(m \log m)$, otherwise it is $O(nm \log m + n \log n)$, the reason being that the algorithm is now also preprocessing the sensors, and the best sorting algorithm has complexity $O(n \log n)$.

**Algorithm 5** Ordered Sensor-side Greedy

1: sort the sensors in order of decreasing $\max\{p_{ij}\}$
2: $M \leftarrow \{M_1, \ldots, M_m\}$
3: **for** each sensor $S_i$ in arbitrary order **do**
4: $M_k \leftarrow maxarg_{M_j \in M}\{p_{ij}\}$, such that
$e_{ik}$ + total utility cumulated by $M_k \leq d_k$
5: **if** $M_k \neq \emptyset$ **then**
6: assign $S_i$ to $M_k$
7: **end if**
8: **end for**

## 4. PERFORMANCE EVALUATION

To evaluate the performance of these algorithms we tested them on randomly generated problem instances using the simulation environment implemented in Java.[8] We use two measurements to test their efficiency: the optimality of the solution returned by the algorithms and the running time required to find it. For this reason the experiments that we conducted are divided into two main groups (Section 4.2 and 4.3).

### 4.1 Assumptions

We assume that all the sensors know their geographical locations and hence can determine their utility. The utility offer ($e_{ij}$) of a sensor to a mission is a function of the geographical distance between them: the closer the sensor to the mission, the higher its utility. The utility of a sensor $S_i$ to a mission $M_j$ will be $e_{ij} = \frac{1}{1+D_{ij}^2/c}$ if $D_{ij} \leq SR$, $e_{ij} = 0$ otherwise ($D_{ij}$ is the distance between $S_i$ and $M_j$, and $SR$ is the global sensing range). It models realistic signal attenuation which is relative to the square of the distance from a source. In our experiments $c = 60$ and $SR = 30m$. The utility is 1 if the sensor and the mission lie at the same location. Sensors and missions are generated in uniformly random locations in a 400m $\times$ 400m field, and when sensors are deployed we ensure that the network is connected. Since the algorithms considered here follow a centralized approach, we create also a base station in the field which represents the node where all the information about the network is collected and where all the computation is performed. Each mission has an exponentially distributed demand and priority, so there will be many missions with low demand and low priority and few missions with high demand and high priority.

### 4.2 Optimality Results

We perform three different experiments in which the field size ($400m \times 400m$) is kept constant, but the density of the nodes (i.e. number of sensors) in the field and the number of missions are varied. The number of sensors in the network is incremented for each of the three simulations: 200, 500, and finally 1000 sensors. For each node density, we vary the number of missions from 10 to 150, with an increment of 10 at each step.

The quality of the solution returned corresponds to the value of the objective function in the ILP formulation of SUM described in Section 2 which we call *Total Network Profit*: the higher this value, the better the solution. The Total Network Profit of each algorithm (which is the sum of all the profits $p_{ij}$ of the decided sensor-mission assignment) is compared to the one achieved by the *optimal fractional solution*, which is the solution to the relaxed linear programming formulation of SUM. By relaxed formulation we mean a linear programming formulation of SUM where $x_{ij}$ is real valued.

The Total Network Profit achieved by the optimal fractional solution is computed using *lp_solve*[‡] which is a linear programming solver. We also used lp_solve as an ILP-solver to check if, for the size of our SUM instance, we could obtain an optimal (or at least good) integer solution in a reasonable time. The timing results will prove lp_solve to be computationally expensive in terms of the amount of time taken to produce a solution, as we will see in Section 4.3. Because of this we used a timeout of 10 seconds, after which the Branch-and-Bound algorithm used by lp_solve was stopped. If the algorithm finds any feasible solution within the timeout then it returns the best solution found, that is the one with the highest value for the objective function. If the Branch-and-Bound algorithm does not find any feasible solution before the timeout, it will run again with an increasing timeout and ultimately with relaxed constraints.

In each of the following experimental results we show the average of 10 runs for the Total Network Profit achieved by each algorithm. In the first set of results in Fig. 3 and 4, we show the Total Network Profit achieved by the different algorithms as a percentage of the Total Network Profit achieved by the optimal fractional solution, with a network of 200 sensors. Note that the optimal fractional solution is not always a strict upper bound on our results, in fact the *integrality gap*[§] between the relaxed LP problem and the ILP problem could be very large. With this in mind, we see in Figure 3 that the $(2+\epsilon)$-approximation algorithm (where we chose $\epsilon = 0.005$ because, in our simulation, this was the smallest value allowable to obtain a reasonable computational time), lp_solve and the Ordered Sensor-side Greedy algorithm offer the best solutions. Since in this particular experiment we are using a small number of sensors lp_solve often manages to find a really good solution to the problem, but what is really relevant is that both Ordered Sensor-side Greedy and $(2+\epsilon)$-approximation algorithms have performances comparable to lp_solve. In Fig. 4 we consider only the best three algorithms, and we notice that the difference between the Ordered Sensor-side Greedy solution and $(2+\epsilon)$-approximation algorithm solution is 0.5%-1% in average, which means that they have more or less the same performances in terms of goodness of the solution.

From Figure 5 and 7 we note that lp_solve performances (compared to the Ordered Sensor-side Greedy and the $(2+\epsilon)$-approximation algorithm performances) decrease with the increment of the number of sensors, in fact with 1000 sensors (Fig. 7) the gap between the $(2+\epsilon)$-approximation algorithm and lp_solve is around 3%. Note instead that the gap between Ordered Sensor-side Greedy and $(2+\epsilon)$-approximation algorithm is 2% in average.

Despite the general degrade of the performances of the other approaches compared to the $(2+\epsilon)$-approximation algorithm while increasing the number of sensors, we observe a general improvement on the percentage of the optimal fractional solution achieved by all the algorithms. This is shown in Figure 8 where the maximum percentage of the optimal fractional solution achieved by the $(2+\epsilon)$-approximation algorithm is around 96%, while instead with 200 sensors in Fig. 4 was around 84%.

Figures 7 and 8 show more clearly than the other experiments, that keeping constant the number of sensors while increasing the number of missions leads to a global degradation of the performance. Indeed, the percentage of optimal fractional solution achieved by the $(2+\epsilon)$-approximation algorithm with 10 missions is around 96%, but instead while dealing with 150 missions the performance decreases to 92%.

[‡]http://lpsolve.sourceforge.net/5.5/

[§]If the relaxed solution to an Integer Linear Programming is indicated as $|LPR|$ and the integer solution is $|ILP|$ then: *Integrality Gap* $= \frac{|LPR|}{|ILP|}$.

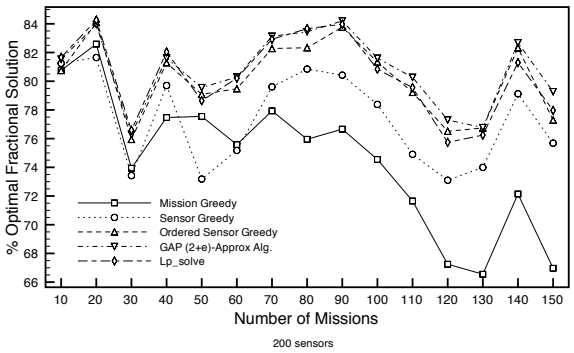


Figure 3: (200 sensors) Total Network Profit of each algorithm.

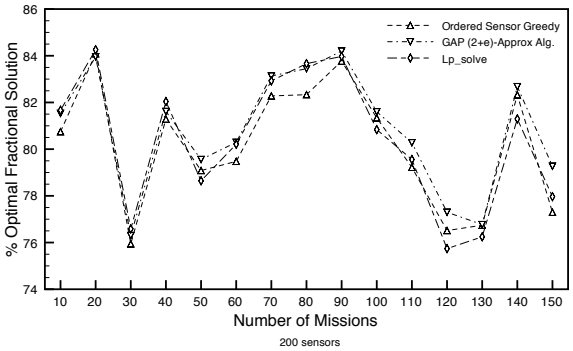


Figure 4: (200 sensors) Total Network Profit of the best 3 algorithms.

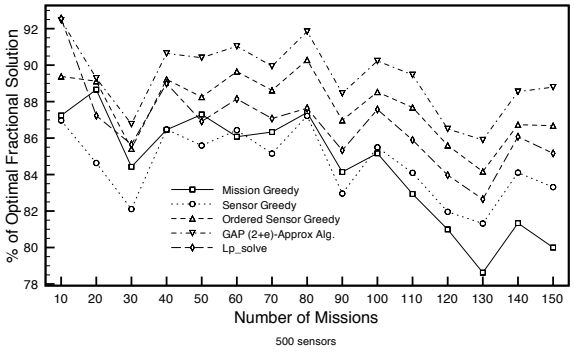


Figure 5: (500 sensors) Total Network Profit of each algorithm.

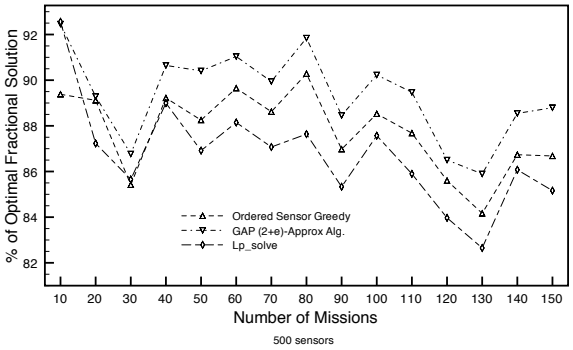


Figure 6: (500 sensors)Total Network Profit of the best 3 algorithms.

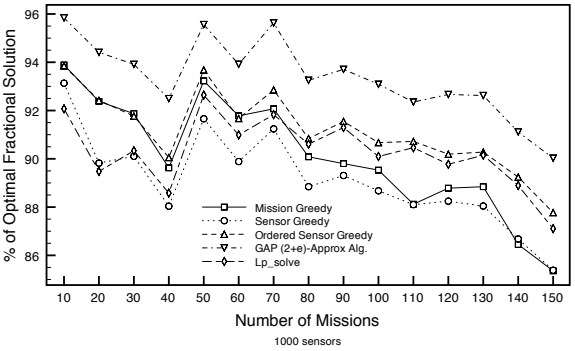


Figure 7: (1000 sensors) Total Network Profit of each algorithm.

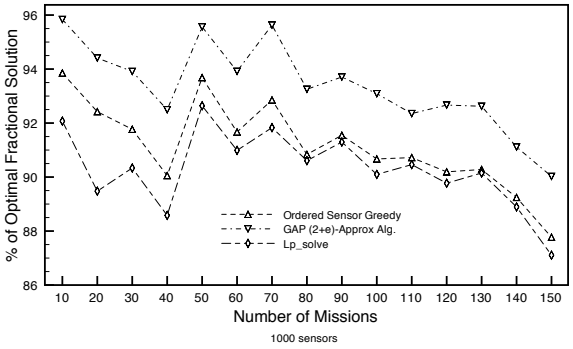


Figure 8: (1000 sensors) Total Network Profit of the best 3 algorithms.

In conclusion, we infer from these results that the best algorithm (between the ones we considered) to solve SUM with an high degree of approximation is the $(2+\epsilon)$-approximation algorithm described in Section 3.1, although we will see in the following Section that it is not the best in terms of effective time spent to compute the solution.

### 4.3 Timing Results

For all the previous experiments we also measured the time required by each algorithm to find a feasible solution to SUM on a Macbook Pro with an Intel Core 2 Duo Processor at 2.4 GHz.

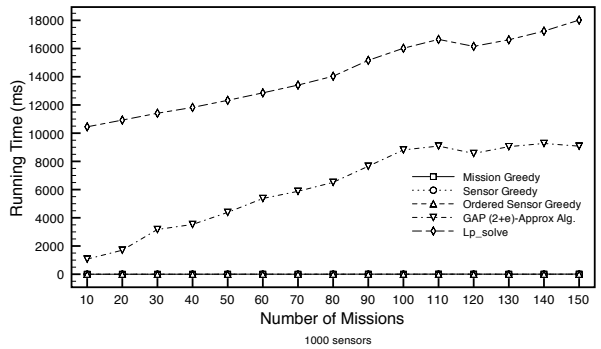


Figure 9: (1000 sensors) Running times of each algorithm.

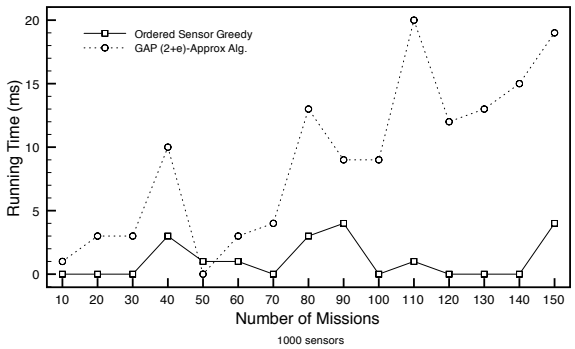


Figure 10: (1000 sensors) Running times of Ordered Sensor side greedy vs (2+$\epsilon$)-approximation algorithm with average $\epsilon = 1.065$.

In Figure 9 we show the timing results of the experiments carried on with a network of 1000 sensors[¶]: lp_solve is the worst among all the approaches as we were expecting; the (2+$\epsilon$)-approximation algorithm is better than lp_solve but worse than all the greedy algorithms. This is due to the fact that the time complexity of the (2+$\epsilon$)-approximation algorithm is $O\left(\frac{mn}{\epsilon^2}\right)$, so if we require a high precision in the solution (i.e. a very small $\epsilon$) there will be a large multiplicative constant ($1/\epsilon^2$). Therefore even if the other greedy algorithms have a worse time complexity, in practice they have better running times compared to the (2+$\epsilon$)-approximation algorithm.

Finally in Figure 10 we show only the running times of the Ordered Sensor-side Greedy algorithm and of the (2+$\epsilon$)-approximation algorithm using an increased value for $\epsilon$ different for every solved SUM instance, which on average was 1.065. We adjusted the value of $\epsilon$ for every solved SUM instance so that the (2+$\epsilon$)-approximation algorithm was able to return a solution with a Total Network Profit very close to that of the solution returned by the Ordered Sensor-side Greedy. We deliberately lowered the performance of the (2+$\epsilon$)-approximation algorithm in terms of quality of the solution returned to check if the running time of the Ordered Sensor-side Greedy remained still the lowest. Indeed Figure 10 proves that also in this situation the Ordered Sensor-side Greedy is the most desirable in terms of running time.

For all these reasons we believe that our novel Ordered Sensor-side Greedy algorithm is a better compromise between optimality of the solution found and effective running time, compared to the (2+$\epsilon$)-approximation algorithm which offers a better solution but at a computational cost which is not desirable. Moreover our choice is reasonable if we consider that the difference between the two Total Network Profits returned by these two algorithms is only 1%, proving that their performances in terms of quality of the solution returned are very similar.

## 5. RELATED WORK

There has been a lot of work recently on the sensor-mission assignment problem. For example, Byers et al[9] approach the assignment problem using the notions of utility and cost. They find a solution that maximizes the utility while not exceeding a predefined budget, using a cost model based on energy consumption. In our SUM model we do not distinguish between cost and utility; moreover we consider multiple missions with different priorities that could possibly compete for the same subset of sensors, where instead Byers assumes that tasks are all equally important.

A problem that is strongly correlated to SUM is presented in Bar-Noy et al[7] and Rowaihy et al,[3] which is called the Semi-Matching with Demands (SMD) problem. It models the problem of assigning sensors to multiple competing missions as a bipartite graph where nodes can be sensors or missions. Each mission is associated with a demand value, that is the utility demand required by the mission to be satisfied, and a profit value, that

[¶]The timing results with 200 and 500 sensors have the same trend of the results in Figure 9.

represents the importance or priority of the mission; each sensor-mission pair is associated with a utility offer (possibly 0).

The similarities with the definition of the SUM problem are evident but there are also many important differences. First of all, SUM gives a different meaning to the *demand* of a certain mission, i.e. it is intended as an upper bound on the utility requested by a mission, since the utility demanded by a mission is usually very uncertain and we don't want to waste sensing resources, as we noticed in Section 1. The most important difference with our work is that in SMD the goal is to find a sensor assignment which maximizes the profits of the satisfied missions, with no credit for partially satisfied missions. Instead, the SUM goal is to assign sensors to missions for which they are most useful, without caring about the particular satisfaction of each particular mission. SUM considers only the "global happiness" of the set of mission and not of every single mission, in such a way that there is no waste of information gathered by the network.

Another thing to observe is that SMD model gives much more importance to the priority (or importance) value of every single mission, since it uses this parameter to decide which are the first missions that have to be satisfied. SUM instead uses this parameter only to scale the utility that the sensor will bring to the mission, as we described in Section 2. The SUM model is therefore to be preferred over the SMD model when the importance of a mission is very uncertain. The priority (i.e. importance) of a single mission is usually not exactly known, because we should have an absolute scale to express the importance of a mission in comparison with other mission priorities to determine it exactly, but for example in a coalition context this is not always true.

SUM can also be viewed as a generalization of a model developed for the sensor deployment problem, introduced in Pizzocaro et al.[10] The sensor deployment problem is solved in two steps: first, sensors are assigned to zones, and finally sensors are deployed inside each zones to which they were assigned. The first subproblem is modeled as an extension of the multiple knapsack problem which can be considered the base for the model presented in this paper.

## 6. CONCLUSION

In this paper, we considered an homogeneous sensor network (e.g. only video sensors all with the same capabilities) where sensing resources cannot be shared by different sensing tasks (or missions). We defined the Sensor Utility Maximization (SUM) problem, where the goal is to maximize the total network profit, i.e. to find an assignment of sensors to missions that maximizes the total benefit for the sensor network by sending the utility where it is most helpful. We analyzed its complexity, showing that is NP-Complete and a special case of the Generalized Assignment Problem (GAP). Furthermore we looked at several pre-existent centralized algorithms which we adapted to solve the SUM problem and we also developed a novel greedy algorithm. Our simulation results show that our greedy algorithm has performance comparable to the state-of-the-art algorithms developed to solve GAP and appears to offer the best trade-off between quality of solution and computation cost.

## ACKNOWLEDGMENTS

This research was sponsored by the U.S.Army Research Laboratory and the U.K. Ministry of Defence and was accomplished under Agreement Number W911NF- 06-3-0001. The views and conclusions contained in this document are those of the author(s) and should not be interpreted as representing the official policies, either expressedor implied, of the U.S. Army Research Laboratory, the U.S. Government, the U.K. Ministry of Defence or the U.K. Government. The U.S. and U.K. Governments are authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation hereon.

## REFERENCES

[1] "Jp 2-01 joint and national intelligence support to military operations." Website (Last accessed: July 25, 2008). http://www.dtic.mil/doctrine/jel/new_pubs/jp2_01print.pdf, pages III-10-11.

[2] Ibarra, O. H. and Kim, C. E., "Fast approximation algorithms for the knapsack and sum of subset problems," *J. ACM* **22**(4), 463–468 (1975).

[3] Rowaihy, H., Johnson, M., Bar-Noy, A., Brown, T., and La Porta, T., "Assigning sensors to competing missions," in [*Proceedings of the IEEE Globecom 2008*], (2008). to appear.

[4] Khuri, S., Bäck, T., and Heitkötter, J., "The zero/one multiple knapsack problem and genetic algorithms," *Proceedings of the 1994 ACM symposium on Applied computing* , 188–193 (1994).

[5] Garey, M. and Johnson, D., [*Computers and Intractability: A Guide to the Theory of NP-Completeness*], WH Freeman & Co. New York, NY, USA (1979).

[6] Cohen, R., Katzir, L., and Raz, D., "An efficient approximation for the generalized assignment problem," *Inf. Process. Lett.* **100**(4), 162–166 (2006).

[7] Bar-Noy, A., Brown, T., Johnson, M., LaPorta, T., Liu, O., and Rowaihy., H., "Assigning sensors to missions with demands.," in [*ALGOSENSORS.*], (2007).

[8] Johnson, M. P., Rowaihy, H., Pizzocaro, D., Bar-Noy, A., Chalmers, S., La Porta, T., and Preece, A. D., "Frugal sensor assignment," in [*Proceedings of 4th IEEE International Conference on Distributed Computing in Sensor Systems (DCOSS '08)*], (2008).

[9] Byers, J. and Nasser, G., "Utility-based decision-making in wireless sensor networks," Tech. Rep. 2000-014 (1 2000).

[10] Pizzocaro, D., Chalmers, S., and Preece, A., "Sensor assignment in virtual environments using constraint programming," in [*Applications and Innovations in Intelligent Systems XV: Proceedings of AI-2007, the Twenty-seventh SGAI International Conference on Innovative Techniques and Applications of Artificial Intelligence*], Springer-Verlag, Cambridge, UK (December 2007).

## APPENDIX A. IMPLEMENTATION OF GAP AND KNAPSACK ALGORITHMS

As a matter of note, the Knapsack algorithm requires a lot of memory, since it generates two matrices (one to compute the objective function and the other to backtrack the solution) of dimension $n \times U$, where both $n$ and $U$ are very large. The number of items $n$ is the number of sensors generated and in our experiments this is a very large number (since we want to be able to deal with very big sensor networks in their thousands). The upper bound $U$ is large since in the classic implementation of the Knapsack algorithm this upper bound is computed by: $U := \tilde{p} \cdot n$; such an upper bound is not beneficial, especially if we give as input to the knapsack algorithm a large number of items and the knapsack has a small capacity (exactly the case of the SUM problem, since missions requires on average a utility computed by assigning three or at most four sensors to the mission). The fact that the dimensions of the matrices are very large also has strong consequences on the effective computational time of the Knapsack algorithm, since it will have to fill a large matrix to find the solution. We solved this problem of *memory usage* and of *computational time* using three techniques. The first technique is to reduce the number of items $n$ given in input at each round of the GAP algorithm to the Knapsack subroutine. We delete from the input those items that have zero profit for that specific bin $j$, and also items that are too large to fit into the bin (i.e. in terms of the SUM problem $e_{ij} > d_j$, and in terms of the Knapsack problem $w_i > c$). In this way we obtain a reduction of $n$, that in our specific test cases are from thousands to tens of sensors; drastically reducing the number of matrice rows created by the Knapsack subroutine.

The second technique applied was reduce the number of columns of the matrices used in the Knapsack subroutine ($U$). We first solve the knapsack problem using the FPTAS with $U = \sum_i p_i$ and without creating the matrix of pointers. We then find the value of $|ALG|$ (the optimal value of the objective function) (as described in Section 3.1). This computation requires much less memory, since we use a better upper bound and we do not create the matrix of pointers. Finally we rerun the Knapsack subroutine with $U = |ALG|$ and this time generate the matrix of pointers. Therefore the algorithms will generate two matrices with the minimum number of columns. These two techniques improve both memory usage and effective computational time, since they reduce the dimensions of the matrices used. Simply stated they limit the search space by reducing the number of possible states.

The third technique improves the memory usage but not the effective time spent to compute the solution in the Knapsack subroutine. It consists of substituting the $n \times U$ matrix used to compute the value of the objective function with a $2 \times U$ matrix (the matrix of pointers remains $n \times U$). This idea derives from the dynamic programming implementation of the knapsack algorithm (Algorithm 2), requiring only the value contained in the previous row $l - 1$ to compute the value of row $l$. Making use of this we can generate just a $2 \times U$ matrix, and we can compute the value of the cells of a row by keeping only the previous row and overwriting with the new computed values the current row (that contain older values no longer required).