Complexity and approximation of a geometric local robot assignment problem*

Olaf Bonorden[1] Bastian Degener[2] Barbara Kempkes[1] Peter Pietrzyk[1]

[1]Heinz Nixdorf Institute, Computer Science Departement,
University of Paderborn, 33095 Paderborn, Germany
bono@uni-paderborn.de, barbaras@uni-paderborn.de, toon@uni-paderborn.de

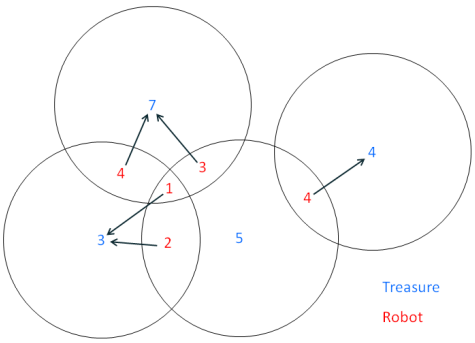
[2]International Graduate School Dynamic Intelligent Systems,
Heinz Nixdorf Institute, Computer Science Departement,
University of Paderborn, 33095 Paderborn, Germany
degener@uni-paderborn.de

June 15, 2009

Technical Report tr–ri–09–299

Abstract

We introduce a geometric multi-robot assignment problem. Robots positioned in a Euclidean space have to be assigned to treasures in such a way that their joint strength is sufficient to unearth a treasure with a given weight. The robots have a limited range and thus can only be assigned to treasures in their proximity. The objective is to unearth as many treasures as possible. We investigate the complexity of several variants of this problem and show whether they are in $\mathcal{P}$ or are $\mathcal{NP}$-complete. Furthermore, we provide a distributed and local constant-factor approximation algorithm using constant-factor resource augmentation for the two-dimensional setting with $\mathcal{O}(\log^* n)$ communication rounds.



*Partially supported by the EU within FP7-ICT-2007-1 under contract no. 215270 (FRONTS), and the DFG-project "Smart Teams" within the SPP 1183 "Organic Computing"

1 Introduction

We consider a scenario where a group of robots is deployed arbitrarily to a Euclidean space, in which treasures are hidden. Being equipped with sensors, the robots are able to detect treasures positioned within a given viewing range. The treasures can have different weights and the robots can have different amounts of strength. The task is to assign the robots to treasures in such a way that the number of treasures which are unearthed by the robots is maximized. A treasure is unearthed if the amount of strength of the robots assigned to it sums up to at least the weight of the treasure. One constraint must be kept for the assignment: A robot may only be assigned to a treasure if the treasure is within the robot's viewing range. The viewing range of all robots is equal. Note that the robots do not actually move before the assignment is calculated. Hence, we are dealing with a static scenario.

Throughout this paper, we will adhere to the figurative description of treasures which have to be unearthed. Nevertheless, in an application these treasures could for example also be tasks which have to be handled by teams of robots. Another application would be the assignment of robots to injured people who have to be rescued from dangerous areas, e.g. sites of catastrophes. These are just a few examples of applications of autonomous robot teams. Our main challenge is that all acting entities have only their own local view and no knowledge about the global state of the environment. This is natural for robot teams deployed to areas unaccessible to humans.

We present an approximation algorithm using *resource augmentation*. This is a well established tool for algorithmic analysis: An algorithm is provided with some additional resources, but is still compared to an optimal algorithm without these.

We distinguish two scenarios of the problem: In the *heterogeneous* scenario the robots can have different amounts of strength, while in the *homogeneous* scenario every single robot has a strength of 1.

Our contribution. We explore the complexity of calculating the optimal solution for the heterogeneous and the homogeneous scenario if all knowledge about the treasures and robots is at hand. Since in many cases the calculation of the optimal solution is impracticable, we present a local, distributed algorithm that uses resource augmentation and provides a constant factor approximation of the optimal solution. The algorithm has a runtime of $\mathcal{O}(\log^* n)$ communication rounds. As a building block, we use an algorithm which might be of independent interest, because it is the first distributed local algorithm for the weighted independent set problem on Δ-bounded degree graphs. It has a Δ^{Δ}-approximation ratio, which is constant in our case.

Related work. This paper deals with allocating robots to points in a Euclidean space. A related area is the one of Multi-Robot Coordination, especially Multi-Robot Task Allocation (MRTA) (for an overview, see [GM04]). Here, robots need to be allocated to some kind of task, e.g. an emergency handling or exploration task. For each task, every robot has an estimated fitness and performance cost, which depend on the concrete task

and which need not be constant. The utility for a robot to handle one task is often defined as the difference of its fitness and cost. The goal is to assign the robots to the tasks such that all tasks are handled and the overall utility is maximized. Due to the nature of the problems, distributed solutions are often favored. Most approaches to MRTA use experiments or simulations to evaluate the strategies ([LZ05], [BMSS05], [OMS01], [TP07]), but some theoretical results exist as well. In [LBK+05], a two-approximation algorithm is given for allocating exploration tasks to robots. Here, a robot fulfills a task by traveling to a destination point indicated by the task. The cost to be minimized is the overall traveled distance.

For our local approximation algorithm, we need to compute a maximal independent set on unit disk graphs locally. This has just been shown to be possible in optimal time $\mathcal{O}(\log^* n)$ [SW08]. Furthermore, we use a local variant of an approximation algorithm for the maximal independent set problem on bounded degree graphs [GPS87]. The main idea of this algorithm is to compare the IDs of adjacent nodes and iteratively collapse them. The position of a bit where two IDs differ is concatenated with the value of the bit. Since this shortens the ID exponentially in each iteration, IDs have a constant length after $\mathcal{O}(\log^* n)$ iterations and are all different. After some refinement this yields a maximal independent set. This algorithm has a runtime of $\mathcal{O}(\log\Delta(\Delta^2+\log^* n))$, which is an improvement of a coloring based algorithm given in [Col86] with complexity $\mathcal{O}(\log^* n + 3^\Delta)$. However, it is possible to compute a maximal independent set in only $\mathcal{O}(\Delta^2+\log^* n)$ rounds [Lin92]. Since Δ is bounded by a constant in our case, all algorithms have a runtime of $\mathcal{O}(\log^* n)$. The authors in [SW08] also elaborate on this approach.

We furthermore present a local distributed Δ^Δ approximation algorithm for the weighted maximum independent set problem. The best global approximation algorithm for this problem yields a Δ-approximation [STK03]. Note that we assume that global coordinates are not available to the robots. While our complexity results would still be valid, our algorithm would be a lot simpler and it would run in constant time using the approach of [KMW05] for computing a maximal independent set. They define a global grid, in which generalized unit disk graphs can easily be colored and turned into maximal independent sets in constant time.

Formal problem definition. We define different variants of the UNEARTH TREASURES problem. All of these variants accept the same kind of input, which is modeled by two sets T and R and an integer v. The set $T = \{t_1, \ldots t_n\}$ represents n treasures, the set $R = \{r_1, \ldots r_m\}$ represents m robots. The viewing range of the robots is represented by v. Since v is equal for all robots, exactly all robots positioned in the sphere around a treasure of radius v can be assigned to the treasure. We call this the viewing range v of the treasure. With every treasure $t_i \in T$ and every robot $r_j \in R$ a position $p(t_i)$ respectively $p(r_j)$ in the d-dimensional Euclidean space is associated. Each of the $n+m$ positions is modeled as a d-tuple. Additionally, we associate an integer value $w(t_i)$ with treasure t_i representing its weight and an integer value $s(r_i)$ with robot r_i representing its strength. The function $assign : R \to T$ is defined in such a way that for all $r_i \in R$ the implication $(assign(r_i) = t_j \Rightarrow \|p(r_i) - p(t_j)\| \leq v)$ is true. This means that it only as-

signs a robot to a treasure, if the treasure is in the robot's viewing range. The function $unearth : (T,R,assign) \rightarrow \mathbb{N}$ computes the number of treasures that can be unearthed if the assignment *assign* is employed. A treasure t_j counts as being unearthed under *assign* if $\sum_{i|assign(r_i)=t_j} s(r_i) \geq w(t_j)$.

We define the decision problem UNEARTH TREASURES by a function $k : \mathbb{N} \rightarrow \mathbb{R}$ with $k(n) \leq n$. This function tells us how many of the n treasures are supposed to be unearthed. In general, the decision problem of UNEARTH TREASURES with the function $k(n)$ is formulated as follows: Given two arbitrary sets T and R and an integer v as defined above, does a function *assign* exist with $unearth(T,R,assign) \geq k(n)$? We only consider k which are computable in polynomial time and which are monotonically increasing. In Section 3 we consider the optimization variant of the problem.

Note that we formally classify a scenario as homogeneous, if for all $r_i \in R$ the equality $s(r_i) = 1$ is true. Otherwise, we call it heterogeneous.

Organization of the paper. In Section 2 we deal with the UNEARTH TREASURES problem in the heterogeneous and the homogeneous scenario. For both scenarios, we prove the complexity of different variants of the problem, i.e. for different functions $k(n)$. Then we present a local approximation algorithm using resource augmentation in Section 3. We conclude in Section 4 by discussing some open problems.

2 The complexity of robot assignment

This section deals with the complexity of finding a solution for the different variants of the UNEARTH TREASURES problem. In order to keep the results as general as possible, we consider a centralized version in two dimensions. In Subsection (2.1), the problem is analyzed within the more general heterogeneous scenario, while in Subsection (2.2) we investigate whether a restriction of the strength-values of the robots to 1 simplifies the problem. Table 1 shows an overview of the results presented in this section.

2.1 The heterogeneous scenario

In this section we will analyze four variants of the UNEARTH TREASURES problem in the heterogeneous scenario. First we show that providing a solution when just one treasure is supposed to be unearthed ($k(n) = 1$) is in $\mathcal{P}$. Then, for $k(n) = c$ with a constant $c > 1$ (Can a constant number of treasures be unearthed?), we show that the problem becomes weak $\mathcal{NP}$-complete. The third variant we analyze is $k(n) = n$, i.e. the question if all treasures can be unearthed. We show this variant to be strong $\mathcal{NP}$-complete. Finally, we show that the problem stays strong $\mathcal{NP}$-complete even if we relax the constrains from $k(n) = n$ to $k(n) \in \Omega(n^{\varepsilon})$ and $0 < \varepsilon \leq 1$ for each $k(n)$.

2.1.1 Unearthing a single treasure $(k(n) = 1)$

Theorem 1. *The variant of* UNEARTH TREASURES *with* $k(n) = 1$ *in the heterogeneous scenario is in* $\mathcal{P}$.

$k(n)$	homogen.	proof sketch
general	strong $\mathcal{NP}$-complete	reduction from PL. INDEP. SET
1	$\mathcal{O}(n \cdot m)$	
c	$\mathcal{O}(n^c \cdot m^3)$	multiple net flows
$[c, n^\varepsilon]$	unknown	
$[n^\varepsilon, n-n^\varepsilon]$	strong $\mathcal{NP}$-complete	add treasures, reduction from general $k(n)$
$[n-n^\varepsilon, n-c]$	unknown	
$n-c$	$\mathcal{O}(n^c(n+m)^3)$	multiple net flows
n	$\mathcal{O}((n+m)^3)$	netflow
$k(n)$	heterogeneous	proof sketch
general	strong $\mathcal{NP}$-complete	see $k(n)=n$
1	$\mathcal{O}(n \cdot m)$	
c	weak $\mathcal{NP}$-complete $\mathcal{O}((n \cdot m \cdot \text{max str.})^c)$	reduction from PARTITION dynamic program
$[c, n^\varepsilon]$	$\mathcal{NP}$-complete	reduction from PARTITION
$[n^\varepsilon, n-n^\varepsilon]$	strong $\mathcal{NP}$-complete	add treasures, reduction from $k(n)=n$
$[n-n^\varepsilon, n-c]$	strong $\mathcal{NP}$-complete	see $[n^\varepsilon, n-n^\varepsilon]$
$n-c$	strong $\mathcal{NP}$-complete	see $[n^\varepsilon, n-n^\varepsilon]$
n	strong $\mathcal{NP}$-complete	reduction from 3-SAT

Table 1: Complexity Classes

Proof. It is easy to check for each treasure one after the other whether it is satisfied or not by assigning all adjacent robots to this treasure. A trivial algorithm runs in time $\mathcal{O}(n \cdot m)$, where n is the number of treasures and m the number of robots. □

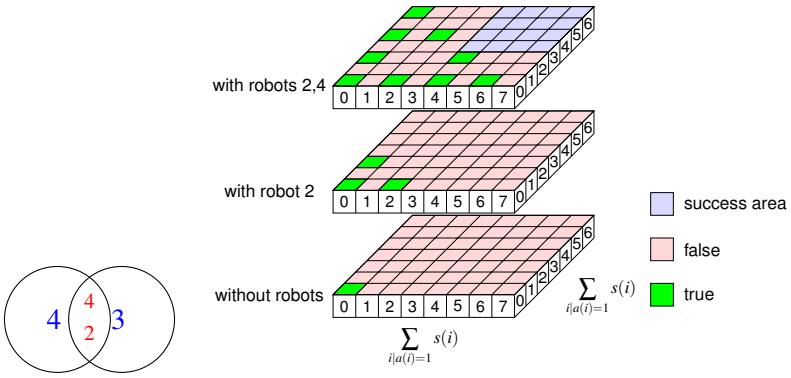
2.1.2 Unearthing a constant number of treasures ($k(n) = c$, $c > 1$)

We show by reduction from PARTITION to UNEARTH TREASURES with $k(n) = c$ that unearthing a constant number of treasures is $\mathcal{NP}$-complete.

Theorem 2. *The variant of* UNEARTH TREASURES *with $k(n) = c$ in the heterogeneous setting is $\mathcal{NP}$-complete for any constant $c > 1$.*

Proof. It is easy to see that all variants of UNEARTH TREASURES are in $\mathcal{NP}$. To show the $\mathcal{NP}$-hardness we reduce the $\mathcal{NP}$-complete PARTITION (see [Kar72]) to UNEARTH TREASURES with $k(n) = 2$. We only need to consider $k(n) = 2$, since deciding if $k(n) = c$ with $c > 2$ treasures can be unearthed is at least as difficult as deciding whether two treasures can be unearthed.

In the PARTITION problem we are given a finite set $A = \{a_1, a_2, \ldots, a_m\}$ of m positive integers which sum up to $2b$, $b \in \mathbb{N}$. Using this set A, we construct an instance of



(a) problem (b) matrix P

Figure 1: Dynamic Program

UNEARTH TREASURES in the following way: We position two treasures t_1 and t_2 with $w(t_1) = w(t_2) = b$ in the plane, so that their viewing ranges intersect. Into this intersection we place m robots $r_1, r_2, \ldots, r_m$ with $s(r_i) = a_i$ for all $1 \leq i \leq m$. An assignment for the robots yields a solution for the PARTITION problem and the reduction takes polynomial time in m. □

Considering the fact that r_i with strength $s(r_i)$ can be interpreted as a team of $s(r_i)$ robots where every robot has strength one and which have to stay together due to a common device, a unary coding of $s(r_i)$ would make sense. Thus, from a practical point of view, it is reasonable to ask whether there exists an pseudo-polynomial algorithm for the UNEARTH TREASURES problem with $k(n) = c$, as is the case for PARTITION. The following theorem provides such an algorithm.

Theorem 3. *There is a pseudo-polynomial time algorithm for the* UNEARTH TREASURES *problem with* $k(n) = c$.

Proof. We use a dynamic programming technique which is similar to the one usually used to solve the PARTITION problem.

As a first step, we identify the treasures that are candidates for being satisfied in a final solution. There are $\binom{n}{c}$ possible combinations. Since c is a constant and $\binom{n}{c} < \frac{n^c}{c!}$, the number of possible combinations is polynomial in n. We define a dynamic program to deal with each combination.

The dynamic program receives c treasures $t_1, \ldots t_c$ and the robot set $R = \{r_1, \ldots, r_m\}$ as input. It iteratively creates $m+1$ c-dimensional arrays with boolean entries. We name these arrays $Arr_0, Arr_1, \ldots Arr_m$. First, we give a definition for array Arr_i $(0 \leq i \leq m)$ and then describe an efficient way to compute Arr_i using Arr_{i-1}.

The array Arr_i is defined with the help of the set $R_i := R_{i-1} \bigcup \{r_i\}$ with $R_0 := \emptyset$. The dimension j $(1 \leq j \leq c)$ of Arr_i represents the treasure t_j. Every dimension of every array has length $\sum_{l=1}^{m} s(r_l)$. The value of the entry $Arr_i[x_1, x_2, \ldots, x_c]$ is only set to *true* if a function $assign : R_i \rightarrow \{t_1, \ldots t_c\}$ (as defined in Section 1) exists, such that for all t_j $(1 \leq j \leq c)$ the sum of the strength of all robots assigned to t_j using *assign* is equal to x_j.

The values of array Arr_i can be computed easily with the help of array Arr_{i-1}. The entry $Arr_i[x_1,\ldots,x_c]$ is *true*, if the entry $Arr_{i-1}[x_1,\ldots,x_c]$ is *true* or if $Arr_{i-1}[x_1,\ldots,x_k - s(r_i),\ldots,x_c]$ is *true* and r_i is in viewing range of treasure t_k. For Arr_0 only the entry $Arr_0[0,\ldots,0]$ is *true*, while all others are $false$.

If Arr_m contains at least one entry $Arr_m[x_1,x_2,\ldots,x_c]$ with value *true*, such that $w(t_k) \leq x_k$ for all $0 \leq k \leq c$, all c treasures can be unearthed. Figure 1 shows the arrays created by our dynamic problem for a simple instance with two treasures with weight 4 and 3 and two robots with strength 2 and 4.

Our dynamic program has runtime $\mathcal{O}((d \cdot m)^c \cdot m)$ where $d := max(\{s(r_1),\ldots s(r_m)\})$. Since the dynamic program is called a polynomial number of times, the entire algorithm has a polynomial runtime, if d is bounded by a polynomial in $n+m$. □

2.1.3 Unearthing all treasures $(k(n) = n)$

In this section, we show that the problem of unearthing all treasures, contrary to unearthing a constant number of treasures, in the heterogeneous scenario is strongly $\mathcal{NP}$-complete and thus no pseudo-polynomial time algorithm for UNEARTH TREASURES with $k(n) = n$ exists.

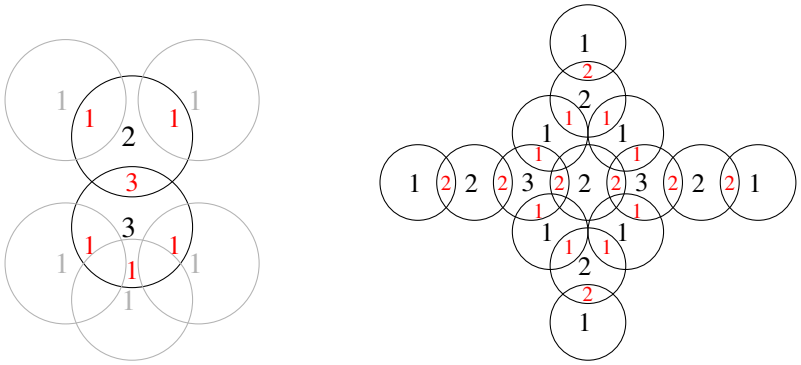
Theorem 4. *The variant of* UNEARTH TREASURES *with* $k(n) = n$ *in the heterogeneous scenario is* strong $\mathcal{NP}$*-complete.*

Proof. It is easy to see that any variant of UNEARTH TREASURES is in $\mathcal{NP}$, thus we just need to prove that it is strong $\mathcal{NP}$-hard. The proof is a reduction from the strongly $\mathcal{NP}$-complete problem 3-SATISFIABILITY (3-SAT). Given a set of clauses we will construct an instance of UNEARTH TREASURES in the Euclidean plane where all treasures can be unearthed if and only if all clauses can be satisfied.

We will describe how to create treasures and robots from the 3-SAT formula and where to place them. To help with the description, we introduce three different constructions: The *clause-gadget*, the *variable-gadget*, and the *connection-gadget*.

Clause-gadget. For every single clause C_k in the 3-SAT formula, one treasure, referred to as the C_k-treasure, with weight 1 is created. Since every clause in 3-SAT consists of exactly three literals, for each C_k-treasure three robots with strength 1 are placed in such a way that they are in range of the C_k-treasure, but not in range of any other treasure. Each of these robots represents one of the clause's literals and is called a (x_i,k)*-literal-robot*, where x_i specifies the literal the robot represents and k the clause it belongs to. We will name a construction consisting of one C_k-treasure and the three corresponding (x_i,k)-literal-robots a C_k*-clause-gadget*. Apparently, after we are finished with constructing the clause-gadgets from a 3-SAT formula containing t clauses we have t treasures and $3t$ robots in the Euclidean plane.

Variable-gadget. The next step in constructing an UNEARTH TREASURES instance from the 3-SAT formula is to create the *variable-gadgets*. For every variable X_i appearing in the 3-SAT formula a single X_i*-variable-gadget* is created. It consists of two



(a) example for an x_i-variable-gadget two times x_i, three times $\bar{x}_i$

(b) crossing

Figure 2: Construction of treasure map for a 3-SAT instance

treasures, referred to as the *x_i-treasure* and the *$\bar{x}_i$-treasure*, and a certain number of robots. These two treasures' positions are chosen in such a way that the two circles defined by their viewing ranges intersect with each other.

Let us assume the literal x_i occurs in p clauses in the SAT formula, while $\bar{x}_i$ occurs in q clauses. This means that we have to set the weight of the x_i-treasure to p and the weight of the $\bar{x}_i$-treasure to q. We also place a single robot in the intersection of the two spheres and set its strength to the maximum of p and q. This robot is referred to as the *X_i-robot*. Next, we place p robots with strength one in range of the x_i-treasure, but out of range of the $\bar{x}_i$-treasure. These robots represent the clauses that contain the literal x_i. We call them *(x_i,k)-variable-robots*, with x_i representing the literal the robot stands for and k specifying the clause. The same is done for the $\bar{x}_i$-treasure: We place q robots, called $(\bar{x}_i,k)$-variable-robots, with strength 1 in its range, but outside of x_i-treasure's range. This concludes the creation of a variable-gadget.

Connection-gadget. A variable-gadget derived from the variable X_i has to be connected to every single clause-gadget that is derived from a clause C_k containing either the literal x_i or $\bar{x}_i$. To create such a connection the *(i,k)-connection-gadget* is introduced, with i referring to X_i and k to C_k. We will first describe a naive idea how to create such a connection. Later we will show how this idea can be implemented in a practical way.

To create the (i,k)-connection-gadget that connects the X_i-variable-gadget and the C_k-clause-gadget, we first have to check whether clause C_k contains the literal x_i or $\bar{x}_i$. (If neither literal x_i, nor $\bar{x}_i$ is contained in C_k, the (i,k)-connection-gadget is not created.) Let C_k contain the literal l, with $l = x_i$ or $l = \bar{x}_i$. We place a treasure with weight 1 in such a way that the (l,k)-literal-robot and the (l,k)-variable-robot are the only robots that can unearth it and call it the *(l,k)-connection-treasure*. Thus, our (i,k)-connection-gadget consists of the (l,k)-literal-robot, the (l,k)-variable-robot and the (l,k)-connection-treasure.

Geometrical constraints can make placing the (l,k)-connection-treasure between the (l,k)-literal-robot and the (l,k)-variable-robot in the way described above impossible. In

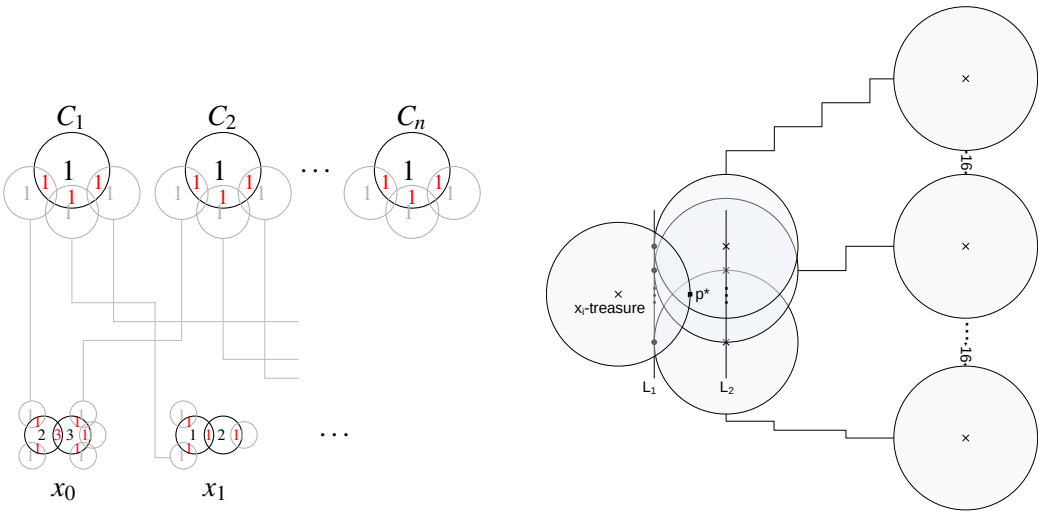
order to still be able to connect a variable-gadget with a clause-gadget, even if those two gadgets are far away from each other, the connection-gadget has to be modified. Instead of placing just one treasure in range of the the (l,k)-literal-robot and the (l,k)-variable-robot, we arrange a chain of treasures with weight 1 between those two robots. This chain's treasures are placed in such a way that the ranges of two neighboring treasures intersect. In every one of these intersection a single robot with strength 1 is placed.

Arrangement of the Gadgets. By now, we have constructed a graph with variable-gadgets and clause-gadgets as nodes and connections-gadgets as edges. Next, we describe the arrangement of these elements. An example for such an arrangement is provided by Figure 3(a): In the upper part of the figure we arrange the clause-gadgets in a row, while the variable-gadgets are also positioned in a row, but on the figure's lower part. Since the connection-gadgets represent our graph's edges, they are displayed as lines connecting the clause- and the variable-gadgets. These lines (that are actually rows of robots and treasures) are placed on a grid with a lattice constant of 16 times viewing range v.

As we cannot guarantee that our construction leads to a planar graph, we also need *crossings* of connection-gadgets for the two dimensional space. A crossing of two connection-gadgets is shown in Figure 3(b). It is important to note that a line (connection-gadget) can only overlap another line on a single point and that the maximal number of lines that can cross each other at a single point of the grid is limited by two. The reason we choose $16v$ for the lattice constant is to have enough space for such crossings. The area covered by the grid that we placed the lines (connection-gadgets) on is quadratic in the number of all the variable-robots from all variable-gadgets. The number of crossings and the longest connection-gadget are also linear.

Geometrically correct construction of a variable-gadget. Assuming that the number of clauses in the 3-SAT formula is t, it might be necessary to connect a variable-gadget to up to t clause-gadgets with the help of up to t connection-gadgets. Creating such connections involves a careful choice of coordinates for the robots and treasures involved in those connections:

Let K be the set of clauses containing x_i and $|K| = s$. Recall that there are s (x_i,k)-variable-robots with $k \in K$ within range of the x_i-treasure. Every (x_i,k)-variable-robot must be positioned in such a way that it is in range of the x_i-treasure and the (x_i,k)-connection-treasure, while not being in range of any other (x_i,l)-connection-treasure with $k \neq l$. It is also important that the coordinates of the robots's position have a size polynomial in s. To achieve this, we need the following construction: Choose an arbitrary position p^* such that the distance between p^* and the x_i-treasure is exactly the treasure's viewing range v. Now, assume that the coordinates of the x_i-treasure are $(0,0)$, while the coordinates of p^* are $(0,v)$. Let L_1 be the line segment between the points $(-\frac{1}{2}v, \frac{1}{2}v)$ and $(\frac{1}{2}v, \frac{1}{2}v)$ and L_2 the line segment between the points $(-\frac{1}{2}v, \frac{3}{2}v)$ and $(\frac{1}{2}v, \frac{3}{2}v)$. We now place the variable-robots on L_1 in such a way that the distance between two of these robots is at least $\frac{s}{v}$. The connection treasures are placed on L_2 in the same way. Additionally (x_i,k)-connection-treasure is placed in such a way that it has minimal distance to the



(a) Construction of treasure map for a 3-SAT instance: example for $(x_0 \vee x_1 \vee x_i) \wedge (\bar{x_0} \vee \ldots) \wedge \ldots$

(b) Arrangement of variable treasures and the corresponding connection gadgets

Figure 3: Positioning of treasures in the plane

(x_i,k)-variable-robot. See also figure 3 (b).

Satisfiability vs. unearthing treasures. In this paragraph we show that all treasures in an instance of UNEARTH TREASURES created from a 3-SAT formula according to the instructions presented this far can only be unearthed if and only if all clauses from the 3-SAT formula can be satisfied.

Consider a 3-SAT formula and an assignment A of TRUE/FALSE values for the formula's variables. We create an UNEARTH TREASURE instance from the formula as described above and assign the robots to the treasures according to the assignment A. If the variable X_i is set to TRUE by A, the X_i-robot is assigned to unearth the x_i-treasure. In the case that A assigns FALSE to X_i, the X_i-robot is sent to unearth the $\bar{x}_i$-treasure. The assignment for the rest of the robots follows directly from the X_i-robots' assignment. We will explain the assignment for the $X_i = true$ case (the $X_i = false$ is analogous) explicitly:

Assigning the X_i-robot to the x_i-treasure allows all (x_i,k)-variable-robots (with $k \in 1,\ldots,n : x_i \in C_k$) to unearth their (x_i,k)-connection-treasures, while the $(\bar{x}_i,l)$-variable-robots (with $l \in 1,\ldots,n : \bar{x}_i \in C_l$) are forced to team up together and unearth the $\bar{x}_i$-treasure. Since every (x_i,k)-variable-robot was free to unearth its connection-treasure, the (x_i,k)-literal-robots are free to unearth their corresponding C_k-treasures and thus satisfy the C_k clauses containing the literal x_i. On the other hand the $(\bar{x}_i,l)$-literal-robots are needed to unearth all $(\bar{x}_i,l)$-connection-treasures, and thus the are not available to unearth the C_l-treasures containing the $\bar{x}_i$ literal. This means that the the C_l-clauses are not satisfied by the $\bar{x}_i$ literal and that other literals are needed to satisfy them. □

2.1.4 Unearthing a polynomial number of treasures $(k(n) \in \Omega(n^\varepsilon), (0 < \varepsilon \leq 1))$

In this section we show that the problem remains strong $\mathcal{NP}$-complete for a broad range of parameters. This is a straight forward generalization of the previous sections, where we considered some special cases. Now, we are not dealing with a single problem, but we are actually dealing with an infinite set of problems. More formally, we prove that the following theorem holds for every fixed ε:

Theorem 5. *The variant of* UNEARTH TREASURES *with fixed* $k(n) \in \Omega(n^\varepsilon), (0 < \varepsilon \leq 1)$ *in the heterogeneous scenario is strong* $\mathcal{NP}$*-complete.*

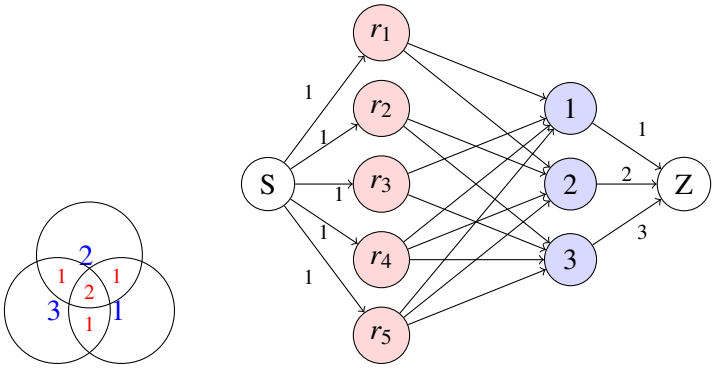
Proof. We reduce the UNEARTH TREASURES problem with $k(n) = n$ to the variant where only $f(n) \in \Omega(n^\varepsilon)$ treasures have to be unearthed. So we are given an instance I of UNEARTH TREASURES with n treasures and are asked whether it is possible to unearth $k(n) = n$ of them. For a constant $0 < \varepsilon \leq 1$, we create an instance $\bar{I}$ with $\bar{n}$ treasures from I such that iff it is possible to unearth $f(\bar{n})$ of the treasures in $\bar{I}$ it is possible to unearth $k(n) = n$ of the treasures in I. To achieve this, we extend I with t treasures which cannot be reached by any robot and therefore cannot be unearthed. This means that in $\bar{I}$ we have $\bar{n} = n + t$ robots. If we choose t in a way that $f(\bar{n}) = k(n)$ and therefore $f(n+t) = n$, $f(\bar{n})$ treasures in $\bar{I}$ can be unearthed if and only if $f(n)$ treasures can be unearthed in I. Moreover, t must be greater than or equal to 0 (we cannot add a negative number of treasures) and polynomial in n, because otherwise the reduction would not be polynomial (since f is computable in polynomial time, this also holds for a t polynomial in n). So now we still need to prove that such a t exists.

There exists a t' such that $f(n+t) = f(n) + t'$, where $t \geq 0$ if and only if $t' \geq 0$, because f is monotonically increasing. Since $f(n) \leq n$ (we cannot unearth more than all treasures) and $f(n) + t' = k(n) = n$, $t' \geq 0$ and therefore also $t \geq 0$. Moreover, because $f(n) \geq 0$, $t' \leq n$. To see that t is polynomial in n, note that $f(n+t) \geq c \cdot (n+t)^\varepsilon$ for large n and a constant c, because $f(n) \in \Omega(n^\varepsilon)$. Additionally, $f(n+t) = f(n) + t' \leq 2n$. Putting these two inequalities together, $c \cdot (n+t)^\varepsilon \leq 2n$ and therefore $t \leq (\frac{2n}{c})^{\frac{1}{\varepsilon}} - n$. Since c and ε are constants, it follows that t is polynomial in n. □

2.2 The homogeneous scenario

In the homogeneous scenario, all robots have the same strength value set to 1. In this section we analyze whether adding this constraint simplifies the problem in some of the variants for the function k. This is the case: The problem is in $\mathcal{P}$ if $k(n)$ or $n - k(n)$ is constant (where the case $k(n) = 1$ follows directly from the heterogeneous setting, but this is not the case for other c). It is strongly $\mathcal{NP}$-complete for general k and for every k with $k(n) \in \Omega(n^{\varepsilon_1}) \wedge k(n) \in \mathcal{O}(n^{\varepsilon_2})$, $0 < \varepsilon_1 \leq \varepsilon_2 < 1$.

Note that the homogeneous scenario is equivalent to a scenario where robots are splittable resources and can be allocated to treasures partially. This makes sense if the weight of a treasure corresponds to the energy the robots must spend to unearth it, and robots can help unearthing several treasures until they have no energy left. The equivalence follows from the proof of Theorem 6: If we have an assignment where robots are assigned



(a) problem instance (b) network for flow algorithm

Figure 4: Example for Flow Algorithm

partially to treasures, we can build a flow network like shown in the proof omitting all non-satisfied treasures. The assignment corresponds to a maximum flow in the network with fractional flow on some (assignment) edges. Since all capacities in the network are integer, there also exists an integer maximum flow in the network. This flow also satisfies all treasures (otherwise it would not be a maximum flow) and can be computed in polynomial time.

We now first show the algorithms with polynomial running time, then we prove the $\mathcal{NP}$-hardness claims.

2.2.1 Unearthing all treasures $(k(n) = n)$

Compared to the heterogeneous setting, the situation changes in this scenario. There is a polynomial time algorithm to solve the problem. We will model an instance of the robot problem as a flow network. It is well-known that standard flow network problems can be solved in polynomial time.

Theorem 6. *The variant of* UNEARTH TREASURES *with $k(n) = n$ (can all treasures be unearthed?) in the homogeneous setting is in $\mathcal{P}$.*

Proof. We model an instance of the robot problem for $k(n) = n$ as a flow network. There is one source node and one sink node. Additionally, we create one node for each robot and one for each treasure. We have an edge from the source node to each robot node with an upper bound of 1 for the flow on this edge. We call these edges *robot edges*. There is also an edge from each robot node to those treasure nodes which are reachable from the respective robot. These edges are called *assignment edges*. From each treasure node we create an edge to the sink. Here, the capacity of the edge is set to the weight of the treasure to be unearthed. We call these edges *treasure edges*. See figure 4 for an illustration.

Since standard flow problems with integer capacities can be solved in polynomial time, the maximum flow f of this network can also be computed in polynomial time.

Moreover, a maximum flow is integer on each edge [CLRS01]. Due to the capacities on the robot edges, in a maximum flow there is at most one assignment edge from each robot node with a flow of 1, while all other assignment edges have a flow of 0. So the flow on the assignment edges corresponds to an assignment of robots to treasures. All treasures can be unearthed if and only if in a maximum flow all treasure edges are saturated, which can be checked in polynomial time. □

2.2.2 Unearthing all but a constant number of treasures ($k(n) = n - c$)

The formulation of the problem for $k(n) = n$ does not help to solve the problem for general $k(n)$. However, deciding whether all but c treasures can be satisfied for a constant c can be solved by a simple extension.

Theorem 7. *The variant of* UNEARTH TREASURES *with $k(n) = n - c$ (can all but c treasures be unearthed?) in the homogeneous setting is in $\mathcal{P}$ for all $c \geq 0$.*

Proof. The problem is equivalent to the problem with $k(n) = n$ if c treasures are removed from the problem including those which are not unearthed in an optimal solution. For any $k(n) = n - c$ there are $\binom{n}{c}$ possible combinations of treasures that can be removed. Since $\binom{n}{c}$ is polynomial in n for any constant c, only a polynomial number of networks with a polynomial runtime each has to be considered, resulting in an algorithm running in polynomial time. □

Note that the technique used above does not result in a polynomial time algorithm for any c larger than a constant. That is due to the fact that $\binom{n}{f(n)}$ is larger than any polynomial as n tends to infinity if $f(n)$ is larger than a constant and smaller than $\frac{n}{2}$. (If $f(n)$ is larger than $\frac{n}{2}$, we refer to section 2.2.3).

Lemma 1. *For $c < f(n) < \frac{n}{2}$ and a constant c, the following holds: $\lim_{n\to\infty} \binom{n}{f(n)} > p(n)$ for any polynomial p.*

Proof. It is well known that $\binom{n}{f(n)} \geq (\frac{n}{f(n)})^{f(n)}$ holds. We consider two cases: $f(n) \leq \sqrt{n}$ and $f(n) > \sqrt{n}$. For $f(n) \leq \sqrt{n}$ the term $(\frac{n}{f(n)})$ is larger than $n^{\frac{1}{2}}$. If this is raised to any power which is not bounded by a constant, it cannot be a polynom. On the other hand, if $f(n) > \sqrt{n}$ (and is smaller than $\frac{n}{2}$), then $(\frac{n}{f(n)})$ can be bounded from below by 2. Since the exponent is at least $\sqrt{n}$, this results in a super-polynomial function as well. □

2.2.3 Unearthing a constant number of treasures ($k(n) = c$)

The construction in section 2.2.2 suggests that a similar algorithm works for small $k(n)$ as well. One only needs to know which treasures have to be satisfied, remove all other treasures and run the network flow algorithm presented in Section 2.2.1. Again, the number of combinations one has to test is $\binom{n}{c}$ and therefore polynomial. As shown in section 2.2.2, this holds for every constant c. However, for symmetry reasons, this does not work for any $k(n)$ which is a function in n (unless it gets close to n up to a constant again).

2.2.4 Unearthing an arbitrary number of treasures (general $k(n)$)

So far we have only shown that there are special cases in the homogeneous setting which are in $\mathcal{P}$. This was motivated by a special case which is strong $\mathcal{NP}$-complete in the heterogeneous setting. In this section we show that deciding the UNEARTH TREASURES problem with an arbitrary $k(n)$ is also $\mathcal{NP}$-complete in the homogeneous setting. We can assume that the function $k(n)$ is part of the input. This is equivalent to getting an integer k as input and being asked whether at least k treasures can be unearthed.

Theorem 8. *The general variant of* UNEARTH TREASURES *with an integer k as part of the input is strong $\mathcal{NP}$-complete in the homogeneous setting.*

Proof. It is obvious that the problem is in $\mathcal{NP}$, thus we just prove that it is strong $\mathcal{NP}$-hard by reducing the $\mathcal{NP}$-complete problem PLANAR INDEPENDENT SET to it. Given a planar graph G and an integer k', we construct the input for UNEARTH TREASURES in the following way: For each node in G we create a treasure and set its weight to the node's degree, while for each edge e, we create a robot which can only reach the treasures corresponding to the nodes adjacent to e. If these two treasures have to be placed too far apart from each other, we use the construction of a connection-gadget to link them (see Theorem 4). Thus, a set of treasures T and a set of robots R is constructed. To make things easier to explain, we assume that for every connection-gadget only one of its robots is assigned to unearth a treasure corresponding to a node, while the remaining robots unearth the treasures which belong to the connection-gadget. Therefore, unearthing the treasure corresponding to $u \in G$ makes unearthing treasures which correspond to u's neighbor nodes impossible. Thus, unearthing treasures that correspond to a node is the same as building an independent set in G. Let l be the number of treasures in all created connection-gadgets and $k := l + k'$. This means that an algorithm with input T and R which can decide whether k treasures can be unearthed also decides whether an independent set with cardinality k' exists in G.

Note that allowing the robots to behave in a different way than we assumed above (i.e assigning two robots belonging to the same connection-gadget to treasures corresponding to nodes) does not increase the number of treasures that can be unearthed, since for every treasure corresponding to a node that is additionally unearthed due to this behavior, at least one treasure belonging to a connection-gadget cannot be unearthed. □

2.2.5 Unearthing a polynomial number of treasures ($k(n) \in \Omega(n^{\varepsilon_1}), k(n) \in \mathcal{O}(n^{\varepsilon_2}), (0 < \varepsilon_1 \leq \varepsilon_2 < 1)$)

In this section we show that the problem is strong $\mathcal{NP}$-complete for a broad range of parameters, similar to the heterogeneous setting (Section 2.1.4).

Theorem 9. *The variant of* UNEARTH TREASURES *in the homogeneous setting is strong $\mathcal{NP}$-complete for any function k with $k(n) \in \Omega(n^{\varepsilon_1})$ and $k(n) \in \mathcal{O}(n^{\varepsilon_2}), (0 < \varepsilon_1 \leq \varepsilon_2 < 1)$.*

Proof. We first reduce the UNEARTH TREASURES problem with general $k(n)$ to a variant with $k'(n) = \frac{n}{2}$ and then we reduce the variant with $k'(n) = \frac{n}{2}$ to the variant where $f(n)$ treasures have to be unearthed, $f(n) \in \Omega(n^{\varepsilon_1})$ and $f(n) \in \mathcal{O}(n^{\varepsilon_2})$.

For the first reduction, we are given an instance I of UNEARTH TREASURES with n treasures and are asked whether it is possible to unearth $k(n)$ of them. From I, we create an instance $\bar{I}$ with $\bar{n} \geq n$ treasures and ask whether we can unearth $f(\bar{n}) = \frac{\bar{n}}{2}$ treasures.

If $k(n) < \frac{n}{2}$, we add $s = n - 2k(n)$ to the n treasures from I which are independent of the old treasures, and the same number of robots which can unearth exactly these treasures. s is positive and polynomial in n. In the new instance, the question is whether we can unearth $f(\bar{n}) = f(n+s) = f(2n-2k(n)) = n - k(n) = k(n) + s$ treasures. This number of treasures can be unearthed if and only if $k(n)$ treasures in I can be unearthed.

If $k(n) > \frac{n}{2}$, we add $t = 2k(n) - n$ new treasures that cannot be reached by any robot. Again, $t \geq 0$ and polynomial in n, since $k(n) \leq n$. Here, $f(\bar{n}) = f(n+t) = f(2k(n)) = k(n)$. Since $k(n)$ treasures in I can be unearthed if and only if $k(n)$ treasures in $\bar{I}$ can be unearthed, this concludes the first reduction.

For the second reduction, we are given an instance I of UNEARTH TREASURES with n treasures and are asked whether it is possible to unearth $k'(n) = \frac{n}{2}$ of them. For two constants $0 < \varepsilon_1 \leq \varepsilon_2 < 1$, we create an instance $\bar{I}$ with $\bar{n}$ treasures from I such that iff it is possible to unearth $f(\bar{n})$ of the treasures in $\bar{I}$ it is possible to unearth $k'(n) = \frac{n}{2}$ of the treasures in the original instance. To achieve this, we extend the original instance with s treasures and s robots to unearth them and additionally t treasures which cannot be reached by any robot and therefore cannot be unearthed. This means that in our new instance we have $\bar{n} = n + s + t$ robots. If we choose s and t in a way that $f(\bar{n}) = k'(n) + s$ and therefore $f(n+s+t) = \frac{n}{2} + s$, this question is equivalent to the question whether $\frac{n}{2}$ treasures in the original instance can be unearthed. Moreover, s and t must be greater than or equal to 0 (we cannot add a negative number of treasures) and polynomial in n, because otherwise the reduction would not be polynomial. So now we still need to prove that such s and t exist and can be computed in polynomial time:

There exists a t' such that $f(n+s+t) = f(n+s) + t'$, where $t \geq 0$ if and only if $t' \geq 0$, because f is monotonically increasing. Now let $s = (\frac{n}{2})^{\frac{1}{\varepsilon_2}} - n$. $s \geq 0$ iff $n \geq 2^{\frac{1}{1-\varepsilon_2}}$. Moreover, since ε_2 is constant, s is polynomial in n. $f(n+s) + t' = f(n+s+t) = \frac{n}{2} + s = -\frac{n}{2} + (\frac{n}{2})^{\frac{1}{\varepsilon_2}}$. Additionally, since $f(n) \in \mathcal{O}(n^{\varepsilon_2})$, $f(n+s) = f(\frac{n}{2}^{\frac{1}{\varepsilon_2}}) \leq c_2 \frac{n}{2}$ for some constant c_2 and large n. Because $\frac{1}{\varepsilon_2} > 1$, it follows that $t' \geq 0$ and therefore $t \geq 0$ for large n. Moreover, because $f(n+s) \geq 0$, $t' \leq \frac{n}{2} + s = -\frac{n}{2} + (\frac{n}{2})^{\frac{1}{\varepsilon_2}}$.

To see that t is polynomial in n, note that $f(n+s+t) \geq c_1 \cdot (n+s+t)^{\varepsilon_1}$ for large n and a constant c_1, because $f(n) \in \Omega(n^{\varepsilon_1})$. Additionally, because $f(n) \leq n$, $f(n+s+t) = f(n+s) + t' \leq n+s+t'$. Putting these two inequalities together, $c_1 \cdot (n+s+t)^{\varepsilon_1} \leq n+s+t'$ and therefore $t \leq (\frac{n+s+t'}{c_1})^{\frac{1}{\varepsilon_1}} - n - s$. Since c_1 and ε_1 are constants and s and t' polynomial in n, it follows that t is polynomial in n. s can obviously be computed in polynomial time. This is also true for t, because f can be computed in polynomial time. □

3 A local approximation algorithm with resource augmentation

This section describes a local, distributed algorithm which uses resource augmentation and computes a constant factor approximation for the UNEARTH TREASURES problem. In order to be able to execute our algorithm, the robots must be able to perform computations and to communicate with other robots within distance $c \cdot v$, where c is a constant which we will set to 6. Furthermore, every robot has exact information about the position and strength/weight of every robot/treasure within distance $c \cdot v$. The communication is performed in synchronous rounds. In each round, every robot can send an arbitrary number of bits to all robots within distance $c \cdot v$. Only after a round is finished, a robot can react to the information it received in the previous round. Moreover, robots and treasures have a unique ID which can also be communicated to neighbors. We assume that treasures have the same capabilities as robots. (A possible explanation for this is that each treasure was located by a robot which now shares the same position as the treasure and performs its communication.) To avoid trivialities, we demand that each robot has a treasure within its viewing range (to achieve this, a robot with no treasures within distance v switches itself off at the very beginning). The constant factor resource augmentation we use in our algorithm allows the robots and treasures to communicate with each other if the distance between them is at most $c \cdot v$ (not only v), and permits assigning a robot to a treasure if they are within distance $c \cdot v$ of each other (and not just v).

The algorithm we propose consists of two parts. In the first part, local clusters of treasures and robots are built. Then, using resource augmentation as described above, the algorithm computes an assignment of robots to treasures for each cluster. In the second part, the algorithm chooses some clusters and, using the assignments computed in the first part for these chosen clusters, creates an assignment for all robots.

The number of rounds used by our algorithm is $\mathcal{O}(\log^* n)$. Since the runtime in wireless networks is dominated by the time needed for communication and since the computation which robots perform in each round is time efficient, we think that the number of communication rounds is a good measure to determine our algorithm's quality.

Basic Definitions. This paragraph presents definitions which are required in the remaining part of this section.

Given an instance (T,R,v) of the UNEARTH TREASURES problem, we construct the *treasure graph* $\bar{G} = (\bar{V},\bar{E})$ as follows: For each treasure $t_i \in T$ we define a node $v_i \in \bar{V}$. An edge $\{v_i,v_j\} \in \bar{E}$ is created, if the treasures $t_i,t_j \in T$ corresponding to v_i and v_j are within distance $2v$ of each other.

A *valid clustering* of an UNEARTH TREASURES instance is a set of subsets $C_i \subseteq R \cup T$ which we call *clusters* such that

1. each treasure and each robot is in at least one cluster
2. in each cluster C_i, exactly one treasure is marked as cluster-center c_i

Algorithm 3.1 LOCALUNEARTHTREASURES(INPUT: treasures and robots within distance $6v$ of me)

1: $\{$*This algorithm is executed locally by a treasure and described from its point of view*$\}$
2: $\bar{G}_{local}$:= treasure-graph induced by treasures within distance $2v$ of me
3: Use $\bar{G}_{local}$ to compute whether I am in INCLUSION-MAXIMAL INDEPENDENT SET (MIS) of $\bar{G}$
4: **if** NOT in MIS $\{$*I am no cluster-center*$\}$ **then**
5: wait for assignment by cluster-center
6: **else**
7: *myCluster* := robots within distance $3v$ and treasures within distance $2v$ of me
8: T_c := treasures in *myCluster*; R_c := robots in *myCluster*
9: *assignment* := ALLTOALL(T_c, R_c) in *myCluster*
10: *myWeight* := number of unearthed treasures in *assignment*
11: $\widehat{G}_{local}$:= cluster-graph induced by cluster-centers in local $6v$-neighborhood of me with resp. weights
12: *inFinalSet* := LOCALMWIS$(\widehat{G}_{local})$
13: **if** *inFinalSet* **then**
14: tell robots in cluster to use *assignment* (OUTPUT)

3. robots belong to a cluster C_i, iff they are in at most distance $3v$ from the cluster-center c_i

4. treasures belong to a cluster C_i, iff they are in at most distance $2v$ from the cluster-center c_i

5. each cluster-center c_i is contained only in its own cluster C_i.

The *cluster-graph* $\widehat{G} = (\widehat{V}, \widehat{E}, w)$ with $w : \widehat{V} \to \mathbb{N}$ contains one node v_i for each cluster-center c_i of a fixed valid clustering of an instance of UNEARTH TREASURES. Each node v_i has a weight w_i, which is the value of an approximation for UNEARTH TREASURES in its cluster. Iff there is a robot in the considered instance that can reach two cluster-centers c_i and c_j, there is an edge $\{v_i, v_j\}$ in $\widehat{E}$.

Given a graph $G = (V, E)$, a subset V' of V is called *independent*, iff there are no two nodes n_i, n_j in V' such that $(n_i, n_j) \in E$. V' is called an *inclusion-maximal* independent set, iff there is no independent V'' with $V' \subsetneq V''$.

For a graph $G = (V, E)$ and a function *weight* that assigns a real number to each node, a subset V' of V is called *maximum weighted independent set*, iff V' is independent and there is no independent set V'' with $\sum_{v' \in V'} \text{weight}(v') < \sum_{v'' \in V''} \text{weight}(v'')$.

A description of the algorithm. In this paragraph, we provide an intuition for how and why the algorithm works. A formal description is given in Algorithm 3.1. This pseudocode is written from the view of a single treasure, so that the input for the algorithm is the set of robots and treasures which are within the augmented viewing range of the

Algorithm 3.2 ALLTOALL(INPUT: T, R)

```
1: L_1 := treasures sorted by weight, lowest first
2: L_2 := robots sorted by strength, lowest first
3: for all treasures t_i in L_1 do
4:    if a single robot r_j can satisfy t_i then
5:       satisfy t_i by assigning r_j to t_i and delete r_j from L_2 and t_i from L_1
6: while there are treasures in L_1 do
7:    take first treasure t from L_1
8:    while t is not satisfied AND there are robots in L_2 do
9:       assign first robot from L_2 to t
10:      if t is satisfied then
11:         delete t from L_1
12: return assignment of robots to treasures
```

Algorithm 3.3 LOCALMWIS(INPUT: $G_{local} = (V, E, w)$ {*in local neighborhood of me*})

```
1: {This algorithm is executed locally by a node and described from its point of view}
2: while (NOT marked as deleted) and (NOT assigned to MWIS) do
3:    Use G_local to compute whether I am in UNW. INCLUSION-MAXIMAL INDEPENDENT SET(MIS) of G
4:    if NOT in MIS then
5:       wait this round
6:    else
7:       if (for all neighbors in cluster-graph) myWeight > neighbors.myweight then
8:          assign myself to MWIS
9:          mark neighboring nodes as deleted
10:      else
11:         mark myself as deleted
12: return whether assigned to MWIS
```

treasure. At the time of the algorithm's termination, each robot knows the treasure it is assigned to.

In the first step of the algorithm, a valid clustering of the robots and treasures is built. This can be achieved by choosing treasures as cluster-centers and assigning all robots and treasures which are in the corresponding distance of a cluster-center c_i (point 3 and 4 of the definition of a valid clustering) to the cluster C_i. Cluster-centers are chosen by computing an inclusion-maximal independent set on the treasure-graph. Each treasure which is in the independent set marks itself as cluster-center (see lemma 2 for why an inclusion-maximal independent set yields a valid clustering of robots and treasures). Since we have a valid clustering, a treasure and a robot belonging to the same cluster have a Euclidean distance of at most five times the viewing range. This means that if we use resource augmentation to increase the viewing range by a factor of five, we have a special situation in each cluster: Any robot can be assigned to any one treasure. This property is necessary

to use the algorithm ALLTOALL (Algorithm 3.2). This algorithm receives a set of robots and a set of treasures in which all robots can reach all treasures as input. It computes a solution for these two sets, so that the number of treasures unearthed is at least one half of the number of treasures unearthed in an optimal solution (see lemma 4). Now, in each cluster the cluster-center can apply ALLTOALL to compute a solution for its cluster. But having an approximation for each cluster does not directly lead to an approximation for the whole problem instance, since robots can be in more than one cluster and two different cluster-centers might therefore assign these robots to different treasures. Therefore, we select some of the clusters for the final solution, so that we still have a constant factor approximation, but each robot is only in one cluster of the final solution. To do this, we approximate a maximum weighted independent set on the cluster-graph (Algorithm 3.3) and select the clusters in this independent set for the final solution. This computation is efficient (see Lemma 6), because the cluster-graph has a constant degree (see Lemma 3). Ultimately, robots can only belong to one cluster in the final solution and will therefore be assigned to at most one treasure.

Clustering of robots and treasures: Correctness. Now we have a closer look on how the clustering of robots and treasures is computed. The idea is to select treasures as cluster-centers in such a way that a valid clustering is generated and letting all treasures and robots which are in the according distance of a cluster-center belong to its cluster. The next lemma shows how treasures can be selected as cluster-centers.

Lemma 2. *Any inclusion-maximal independent set on the treasure-graph $\bar{G}$ of an instance of* UNEARTH TREASURES *corresponds to a selection of treasures as cluster-centers which yield a valid clustering.*

Proof. Since the clusters are constructed by the choice of cluster-centers, obviously each cluster has a cluster-center. Furthermore, the distances from treasures and robots to cluster-centers are also kept due to the construction of the clusters. We still have to show that each treasure and each robot is in at least one cluster (Proposition 1) and that each cluster-center is only in its own cluster (Proposition 2).

For the first proposition, assume that there is one treasure t which is not in a cluster. It therefore exists no cluster-center within distance $2v$ of t. Therefore, there is no edge between t and a cluster-center in the treasure-graph. So adding t to the independent set increases the size of it and therefore the independent set was not inclusion-maximal. Now assume that there is one robot r which is not in a cluster. It therefore exists no cluster-center within distance $3v$ of r. Since there is at least one treasure in the viewing range v of r, this treasure cannot be within distance $2v$ of a cluster-center. This constitutes the contradiction.

For the second proposition, consider two cluster-centers c_1 and c_2. Since they both are in the inclusion-maximal independent set of the treasure-graph $\bar{G}$, there is no edge $\{c_i, c_j\}$ in $\bar{G}$. This means that c_i and c_j are in distance more than $2v$ of each other and therefore neither of them is in the other's cluster. □

To be able to compute a maximum weighted independent set on the cluster-graph

efficiently, we need the cluster-graph to have a constant degree. Like Lemma 3 states, this is guaranteed if the clustering of robots and treasures is valid.

Lemma 3. *A valid clustering of an* UNEARTH TREASURES *instance results in a cluster-graph with at most degree* $\Delta = 48$.

Proof. Since each cluster-center is only in its own cluster, the distance between two cluster-centers is more than $2v$. This implies that each cluster-center has a circle with a radius v surrounding itself which does not intersect with the according circles of other cluster-centers. We call this circle the *exclusive area* of a cluster-center. Furthermore, two cluster-centers which share an edge in the cluster-graph can be at most in distance $6v$ of each other. Now consider a cluster-center c. All its neighbors in the cluster-graph must be in distance $6v$ around c. This means that the exclusive areas of the neighbors of c are in distance at most $7v$ and therefore in an area of size $\pi \cdot (7v)^2 = \pi \cdot 49v^2$ around c. Since each cluster-center has an exclusive area of size πv^2, there can be no more than $\frac{\pi \cdot 49v^2}{\pi v^2} = 49$ cluster-centers and therefore 48 neighbors of c in this area. □

Choosing clusters for a final solution: Correctness. As soon as the cluster-centers have been defined, each cluster-center computes an approximation for its own cluster. To achieve that each robot can be assigned to each treasure, the viewing range of each robot and treasure is increased by a factor of 6 (5 would be sufficient here, but we need 6 later). The next lemma states that the solution computed by a cluster-center is a two-approximation.

Lemma 4. *Given an instance of* UNEARTH TREASURES *where each robot can reach all treasures,* ALLTOALL *computes a 2-approximation.*

Proof. We compare the solution computed by ALLTOALL with an optimal solution. We show that for each treasure satisfied by ALLTOALL, there is at most one additional treasure satisfied in the optimal solution. In the first loop, each satisfied treasure is satisfied by a single robot. In an optimal solution this robot might have been assigned to another treasure, which is the only one which might become unsatisfiable by this action.

The second loop iterates over all treasures. There are only two possibilities for how the loop can be left. Either all treasures are satisfied, leading to an optimal solution, or there are no robots left to assign. In this case, observe that the treasures with the smallest values are satisfied. We claim that the algorithm assigns at most twice the needed amount of robot strength to each of these treasures, thus the total power of the robots can at most satisfy the double number of treasures in an optimal solution. This follows from the fact that all wasted robot power belongs to a single robot, because no robots are assigned after the treasure is satisfied. Assume for the sake of contradiction that this robot has more power than is needed to satisfy this single treasure. Then it would have been assigned to the treasure in the first loop of the algorithm. □

As soon as the weights of each cluster-center in form of an approximation in each cluster have been calculated, we have to choose the clusters that will be part of the final

solution. This is done by approximating a maximum weighted independent set on the cluster-graph. For neighboring cluster-centers to be able to communicate with each other, it is necessary to increase the viewing range of cluster-centers by a factor of 6 to $6v$.

Lemma 5. *Given a graph G with node weights and a maximum degree Δ,* LocalMWIS *computes a Δ^Δ-approximation of the maximum weighted independent set of G.*

Proof. First of all, we prove that the while-loop terminates after at most $\Delta+1$ rounds, where Δ is the degree of the graph. If a node u is in the inclusion-maximal independent set (MIS) in one round, either u will mark itself or all its neighbors as deleted. If u is not in the MIS, one of its neighbors is. So either u will be marked as deleted by its neighbor or one of its neighbors will be marked as deleted. This leaves no neighbors for deletion after Δ rounds. Furthermore, note that the final set is indeed independent: In each round the nodes that are chosen to the maximal independent set are independent by definition. If they assign themselves to the final set, they mark their neighbors as deleted, which subsequently cannot be assigned to the final set in later rounds.

To show that we get a Δ^Δ-approximation, note that from each node which is not in the final set we can define a path of *dominating nodes* to a node chosen to the final set. A dominating node of a node u is a neighbor of u with a larger weight. For any given node which is not in the final set, the next node on the path of dominating nodes is either the one that marked it as deleted or, if the node marked itself as deleted, a neighboring node with a higher value. This neighbor must exist, since otherwise the node would not have marked itself as deleted. After at most Δ hops from each node, a node in the final set is reached, because there are at most $\Delta+1$ rounds and all remaining nodes in the last round (with no degree) are chosen to the final set. Now consider an arbitrary node u in the final set and all nodes which are not in the final set whose paths lead to u. These are at most Δ^Δ nodes, because this is an upper bound on the number of nodes in a Δ-neighborhood of a Δ-degree graph. All those nodes have a smaller value than u. Therefore the value of the chosen node is at least $\frac{1}{\Delta^\Delta}$ times the value of a set to which all other nodes would have been chosen. This applies to all nodes in the final set. □

Putting it all together. Now we have all preliminaries for proving the correctness of the algorithm.

Theorem 10. *Algorithm* LocalUnearthTreasures *computes a constant-factor approximation of* Unearth Treasures *using factor-6 resource augmentation.*

Proof. According to Lemma 4, the weight of a cluster is a two-approximation of an optimal solution in the cluster. Furthermore, the choice of clusters for a final solution is approximized with a factor of Δ^Δ (Lemma 5), where Δ is the degree of the cluster-graph and therefore constant (Lemma 3). Furthermore, each robot can only be assigned to one treasure within the same cluster and therefore it needs to travel at most distance $5v$. Treasures must be able to communicate with robots and treasures in their own cluster and additionally their neighbors in the cluster-graph. All these robots and treasures are within distance at most $6v$. □

Theorem 10 states the correctness of the algorithm. We still need to analyze its complexity. Note that two different kinds of runtime can be employed. On the one hand, we can count the number of local computations that the robots and treasures perform and on the other hand we can count the communication rounds. In a wireless network, the costly part concerning time and energy consumption is the communication between robots and therefore we use the number of communication rounds for measuring the complexity of our algorithms. This leads to a complexity of $\mathcal{O}(1)$ of the algorithm ALLTOALL, since this algorithm uses no communication besides the final broadcast of the solution. However, it is worth noting that the local computation for ALLTOALL is still efficient in the worst-case: If all treasures and robots are in one cluster, our algorithm takes time $\mathcal{O}(n \log n + m \log m)$.

Lemma 6. *Algorithm* LOCALMWIS *terminates in* $\mathcal{O}(\log^* n)$ *communication rounds.*

Proof. According to the proof of Lemma 5, the while-loop in LOCALMWIS is executed at most $\Delta + 1$ times. In each round of the loop, an inclusion-maximal independent set is computed. This can be done using the algorithm from [GPS87] in $\mathcal{O}(\log^* n)$ rounds. Line 7 can again take Δ time, the remaining of the loop takes constant time. Since Δ is also constant, the combined running time is $\mathcal{O}(\log^* n)$. □

Theorem 11. *Algorithm* LOCALUNEARTHTREASURES *terminates in* $\mathcal{O}(\log^* n)$ *communication rounds.*

Proof. In line 3 of the algorithm, an inclusion-maximal independent set on a unit disk graph is computed. This can be done using [SW08] in $\mathcal{O}(\log^* n)$ communication rounds. Since LOCALMWIS also takes $\mathcal{O}(\log^* n)$ communication rounds due to Lemma 6 and the remaining of the algorithm takes constant time, the overall running time is $\mathcal{O}(\log^* n)$. □

4 Open questions and final comment

Concerning the complexity analysis in Section 2, there are still some functions $k(n)$ left, for which the complexity of the corresponding variant of the UNEARTH TREASURES problem was not covered.

An important characteristic of the local algorithm presented in Section 3 is its use of resource augmentation. A question that arises is, whether resource augmentation is needed to be able to compute a constant factor approximation for the UNEARTH TREASURES problem in polynomial time. If this is the case, it is reasonable to improve the factor for resource augmentation. Even a $(1+\varepsilon)$-augmentation might be possible. This is not the case for the approximation factor: Here, the analysis of the PARTITION problem shows a lower bound of 2.

Another arising question is, how far the constants of our algorithm can be improved. Especially the approximation factor of Δ^Δ for the local weighted maximum independent set problem should be significantly improved, since a Δ-approximation for global algorithms is known. Extensions to higher dimensions seem straight forward.

References

[BMSS05] W. Burgard, M. Moors, C. Stachniss, and F.E. Schneider. Coordinated multi-robot exploration. *Robotics, IEEE Transactions on*, 21(3):376–386, 2005.

[CLRS01] T. H. Cormen, Ch. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*. The MIT Press, 2001.

[Col86] U. Cole, R.and Vishkin. Deterministic coin tossing with applications to optimal parallel list ranking. *Inf. Control*, 70(1):32–53, 1986.

[GM04] B. P. Gerkey and M. J. Mataric. A Formal Analysis and Taxonomy of Task Allocation in Multi-Robot Systems. *The International Journal of Robotics Research*, 23(9):939–954, 2004.

[GPS87] A. Goldberg, S. Plotkin, and G. Shannon. Parallel symmetry-breaking in sparse graphs. In *STOC '87: Proceedings of the nineteenth annual ACM symposium on Theory of computing*, pages 315–324. ACM, 1987.

[Kar72] R. M. Karp. Reducibility Among Combinatorial Problems. In R. E. Miller and J. W. Thatcher, editors, *Complexity of Computer Computations*, pages 85–103. New York: Plenum, 1972.

[KMW05] F. Kuhn, T. Moscibroda, and R. Wattenhofer. On the locality of bounded growth. In *PODC '05: Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, pages 60–68. ACM, 2005.

[LBK+05] M. G. Lagoudakis, M. Berhault, S. Koenig, P. Keskinocak, and A. J. Kleywegt. Simple auctions with performance guarantees for multi-robot task allocation. In *Multi-Robot Systems. From Swarms to Intelligent Automata Volume III*, pages 27–38. Springer Netherlands, 2005.

[Lin92] N. Linial. Locality in distributed graph algorithms. *SIAM J. Comput.*, 21(1):193–201, 1992.

[LZ05] Liu Lin and Zhiqiang Zheng. Combinatorial bids based multi-robot task allocation method. In *Proceedings of the 2005 IEEE International Conference on Robotics and Automation (IRCA)*, pages 1145–1150, 2005.

[OMS01] E.H. Ostergaard, M.J. Mataric, and G.S. Sukhatme. Distributed multi-robot task allocation for emergency handling. In *IEEE/RSJ International Conference on Intelligent Robots and Systems, Proceedings*, volume 2, pages 821–826, 2001.

[STK03] S. Sakai, M. Togasaki, and K.Yamazaki. A note on greedy algorithms for the maximum weighted independent set problem. *Discrete Applied Mathematics*, 126(2-3):313 – 322, 2003.

[SW08] J. Schneider and R. Wattenhofer. A log-star distributed maximal independent set algorithm for growth-bounded graphs. In *Proceedings of the 27th Annual ACM Symposium on Principles of Distributed Computing (PODC)*, 2008.

[TP07] Fang Tang and L.E. Parker. A complete methodology for generating multi-robot task solutions using ASyMTRe-D and market-based task allocation. In *IEEE International Conference on Robotics and Automation*, pages 3351–3358, 2007.