RELOCATION TASKS AND A HIERARCHICAL SUBCLASS

PAVEL SURYNEK[1,2]

[1] Charles University in Prague, Malostranské náměstí 25, 118 00 Praha 1, Czech Republic
[2] Kobe University, 5-1-1 Fukae-minamimachi, Higashinada-ku, Kobe 658-0022, Japan
E-MAIL: pavel.surynek@mff.cuni.cz

Abstract:

A new concept of abstract relocation problem in introduced in this article. It allows to model variety of relocation tasks arising in areas such as mobile robotics, logistics, and computer entertainment. Relocation problems represent a strict generalization of the existent concepts of cooperative path-finding and multi-robot path-planning. An important hierarchical subclass of relocation problem is defined and a solving algorithm is described.

Keywords:

Relocation tasks; Cooperative path-finding; Multi-robot path-planning; Algorithms

1. Introduction and Motivation

This article reports on our findings regarding abstract relocation problems. Our new concept of a relocation problem consists of entities (such as agents, containers, …) arranged in some environment. Entities can interact with each other through special constraints defined in advance. The task is to rearrange entities to some goal arrangement while movements and interactions among entities must satisfy the special constraints along the relocation process (for example agent can move freely; a container can be moved by an agent only – it cannot move by itself; agents and containers should not collide with each other).

Relocation problems represent a strict generalization of cooperative multi-agent path-finding [4], [5], [6]. Consequently it allows modeling of much wider range of real-life problems than the standard cooperative path-finding. Relocation problems can be used to model variety of tasks arising in mobile robotics, logistics, and even in computer entertainment.

Unlike logistic problems known from the classical planning and particularly from the International Planning Competition – IPC [2], relocation problem introduces a strong interaction among entities which makes it computationally challenging [1]. This aspect is also important from the practical point of view as in the real-life we also face strong interactions among objects that we need to model.

In this article we first introduce the new concept of relocation problem formally. Then we define an important subclass of the problem for which we also show a complete solving algorithm that exploits existent path finding algorithms as sub-procedures.

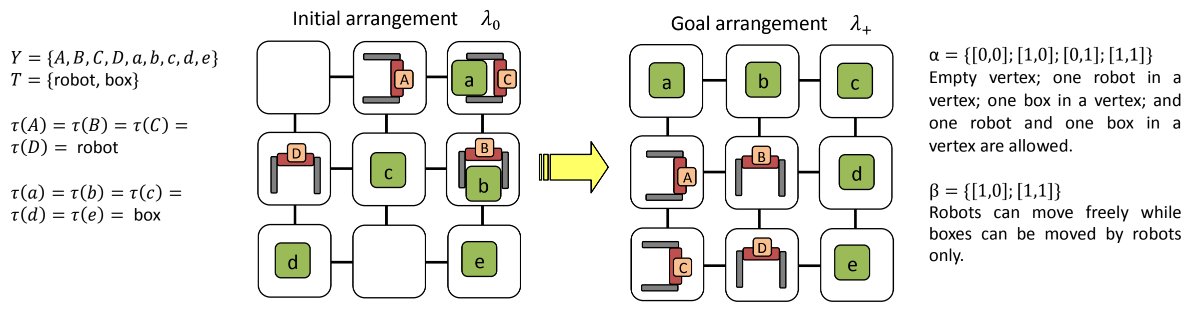
2. Relocation Problem

Let $G = (V, E)$ be an undirected graph where V is a finite set of vertices and $E \subseteq \binom{V}{2}$ is a set of edges. Let $Y = \{y_1, y_2, \dots, y_\mu\}$ with $\mu \in \mathbb{N}$ be a finite set of *entities*. Let $T = \{t_1, t_2, \dots, t_n\}$ with $n \in \mathbb{N}$ be a finite set of entity *types*. Each entity is assigned a type by a function $\tau: Y \longrightarrow T$. A restriction of a given set of entities $Y' \subseteq Y$ on entities of a certain type $t \in T$ will be denoted as $Y'|_t$ and it is defined as follows: $Y'|_t = \{y \in Y | \tau(y) = t\}$.

An arrangement of entities in vertices of the graph G is fully described by a *location* function $\lambda: Y \longrightarrow V$; the interpretation is that an entity $y \in Y$ is located in a vertex $\lambda(y)$. A generalized inverse of λ is denoted as $\lambda^{-1}: V \longrightarrow \mathcal{P}(Y)$; it provides information of a set of entities located in a given vertex. That is, $\lambda^{-1}(v)$ for $v \in V$ is a set of entities located in vertex v with respect to λ. Notice that, $\lambda^{-1}(v)|_t$ for $t \in T$ is a set of entities of type t located in vertex v.

An *allowance constraint* α is an arbitrary relation over an n-tuple of non-negative integers less than or equal to $\mu = |Y|$; that is, $\alpha \subseteq \{0, 1, \dots, \mu\}^n$ (α is a subset of n-ary Cartesian product). Allowance constraints restrict sets of entities that can stay together in the same vertex. More precisely, an arrangement of entities λ is consistent with respect to the allowance constraint α for a vertex $v \in V$ if $[|\lambda^{-1}(v)|_{t_1}|, |\lambda^{-1}(v)|_{t_2}|, \dots, |\lambda^{-1}(v)|_{t_n}|] \in \alpha$. An arrangement of entities λ is called consistent with respect to α if it is consistent for every vertex $v \in V$.

Expectably, we are interested in consistent arrangements only. Consistency of arrangements will be kept along its rearrangements through entity movements. Similarly, a *mobility constraint* β is a relation over an n-tuple of non-negative integers less than or equal to μ; $\beta \subseteq$



$Y = \{A,B,C,D,a,b,c,d,e\}$
$T = \{\text{robot, box}\}$
$\tau(A) = \tau(B) = \tau(C) = \tau(D) =$ robot
$\tau(a) = \tau(b) = \tau(c) = \tau(d) = \tau(e) =$ box
Initial arrangement λ_0
Goal arrangement λ_+
A
B
C
D
a
b
c
d
e
α = {[0,0]; [1,0]; [0,1]; [1,1]}
Empty vertex; one robot in a vertex; one box in a vertex; and one robot and one box in a vertex are allowed.
β = {[1,0]; [1,1]}
Robots can move freely while boxes can be moved by robots only.

Figure 1. An instance of relocation problem. There are two types of entities: robots and boxes. Boxes need to be relocated by robots into desired goal arrangement. This is an instance of a hierarchical subclass of relocation problem.

$\{0,1,\ldots,\mu\}^n$. It defines what sets of entities are movable together. More precisely, a given set of entities $Y' \subseteq Y$ is movable if $[|Y'|_{t_1}|, |Y'|_{t_2}|, \ldots, |Y'|_{t_n}|] \in \beta$.

At each time step, a **single move** of a set of entities from a given vertex to an adjacent vertex can be done. The set of entities that are moved must be movable with respect to β. The allowance constraint α must be preserved in both the source and the destination vertex at the current time step and at the time step following the move. These requirements are expressed formally as follows.

Let λ_i be an arrangement of entities at time step i consistent with respect to α. A set of entities $Y' \subseteq \lambda^{-1}(u)$ with $u \in V$ can be moved to a vertex v such that $\{u, v\} \in E$ if following conditions hold:

(i) $[|Y'|_{t_1}|, |Y'|_{t_2}|, \ldots, |Y'|_{t_n}|] \in \beta$,

(ii) $[|(\lambda_i^{-1}(u) \setminus Y')|_{t_1}|, |(\lambda_i^{-1}(u) \setminus Y')|_{t_2}|, \ldots, |(\lambda_i^{-1}(u) \setminus Y')|_{t_n}|] \in \alpha$, and

(iii) $[|(\lambda_i^{-1}(v) \cup Y')|_{t_1}|, |(\lambda_i^{-1}(v) \cup Y')|_{t_2}|, \ldots, |(\lambda_i^{-1}(v) \cup Y')|_{t_n}|] \in \alpha$.

The above conditions ensure that the resulting arrangement λ_{i+1} at time step $i+1$ is consistent with respect to α again. It holds for λ_{i+1} that $\lambda_{i+1}^{-1}(w) = \lambda_i^{-1}(w)$ for all $w \in V$ other than u and v; $\lambda_{i+1}^{-1}(u) = \lambda_i^{-1}(u) \setminus Y'$; and $\lambda_{i+1}^{-1}(v) = \lambda_i^{-1}(v) \cup Y'$. A move of a set of entities that satisfies above conditions is called an *allowed move* and it is denoted as $Y': u \longrightarrow v$. □

Definition 1 *(Relocation problem)*. A *relocation problem* is a tuple $\Pi = [G = (V,E), Y, T, \tau, \alpha, \beta, \lambda_0, \lambda_+]$ where G is an undirected graph, Y is a finite set of entities, T is a finite set of entity types, τ is an entity type assignment function, α is an allowance constraint, β is a mobility constraints, λ_0 is an initial arrangement of entities consistent with α, and λ_+ is a goal arrangement of entities consistent with α. □

The task is to find a finite sequence of allowed moves of sets of entities such that if it is successively applied on the initial arrangement λ_0, the goal arrangement λ_+ is finally obtained.

Definition 2 *(Solution)*. A *solution* to a relocation problem $\Pi = [G = (V,E), Y, T, \tau, \alpha, \beta, \lambda_0, \lambda_+]$ is a finite sequence $\vec{s}(\Pi) = [Y_1': u_1 \rightarrow v_1, Y_2': u_2 \rightarrow v_2, \ldots, Y_m': u_m \rightarrow v_m]$ with $m \in \mathbb{N}$ of allowed moves where $Y_i' \subseteq Y, u_i \in V$, and $v_i \in V$ for all $j = 1,2,\ldots m$ such that if it is inductively defined that an arrangement λ_i is the result of application of the move $Y_i': u_i \rightarrow v_i$ on the arrangement λ_{i-1} then $\lambda_m = \lambda_+$. The allowance constraint α and the mobility constraint β must be preserved at all the steps. □

An example of typical relocation instance and modeling abilities of relocation problems are illustrated in Figure 1 – at most one robot and at most one box can occupy a vertex while robot and box together are allowed. The box is a passive entity it cannot move by itself it must be moved by a robot. The robot is an active entity which can move either alone or with the box which simulates the process that the box is relocated by the robot.

Notice that, it does not need to hold that $\beta \subseteq \alpha$; that is, a sub-set of entities that is not allowed can be moved. It is because of the fact that moved sub-set can have supporters in the original as well as in the destination vertex with that it forms an allowed sub-set of entities.

When real-life situations taking place in the physical world are modeled as relocation problems it is rarely the case that the number of entities simultaneously occupying the same vertex is unbounded. Observe also, that the space of $\Omega((\mu+1)^n)$ is required to store the allowance and the mobility constraint in the worst case. Bounding the number of entities that can simultaneously occupy the same vertex by a constant $b \in \mathbb{N}$ is hence reasonable and allows us to reduce the worst case space required to store the constraints to $O((b+1)^n)$ (since then it holds that $\alpha, \beta \subseteq \{0,1,\ldots,b\}^n$).

Definition 3 *(Bounded constraints)*. The allowance constraint α is called *b-bounded* for $b \in \mathbb{N}$ if $[c_1, c_2, \ldots, c_n]$ with $c_i \in \mathbb{N}_0$ for $i = 1,2,\ldots,n$ is not satisfied by α whenever $\sum_{i=1}^{n} c_i > b$. Analogically we define b-bounded mobility constraint β. □

A *b-bounded relocation problem* with $b \in \mathbb{N}$ is a relocation problem $\Pi = [G, Y, T, \tau, \alpha, \beta, \lambda_0, \lambda_+]$ where α and β are b-bounded allowance and mobility constraints respectively.

3. Hierarchical Tractable Subclass

Let us now describe an important tractable subclass of the relocation problem. It will be called a *hierarchical* subclass as the movable combinations of entities are arranged in a hierarchical manner by the mobility constraint. It generalizes the existent concept of *cooperative multi-agent path-finding* (Luna, Berkis, 2011; Surynek, 2009).

We will define conditions on a relocation problem instance $\Pi = [G = (V, E), Y, T, \tau, \alpha, \beta, \lambda_0, \lambda_+]$ which makes it a member of the hierarchical subclass. Without loss of generality let us suppose that $G = (V, E)$ is connected since otherwise we can restrict the following reasoning on the individual connected components. Let $T = \{t_1, t_2, \dots, t_\kappa\}$ with $\kappa \in \mathbb{N}$ referring to the depth of the hierarchy be a set of entity types. If the following conditions hold, then Π is called to be κ-hierarchical:

(1) $1 \leq \left|Y|_{t_i}\right| \leq |V| - \xi - 2$ for $i = 1,2,\dots,n$;

(2) $[c_1, c_2, \dots, c_n] \in \alpha$ if $c_i \in \{0,1\}$ for all $i = 1,2,\dots,n$; that is, the allowance constraint α allows at most one entity of the same type in a vertex;

(3) $[c_1, c_2, \dots, c_n] \in \beta$ if $c_i \in \{0,1\}$ for all $i = 1,2,\dots,n$ and $c_j \geq c_{j+1}$ for all $j = 1,2,\dots,n$; that is, $\beta = \{[1,0,\dots,0], [1,1,0,\dots,0], \dots [1,1,\dots,1]\}$.

In other words, mobility constraint permits that an entity of type t_1 can move freely by itself; an entity of type t_2 can move only together with an entity of type t_1 and so on. Generally, an entity of type t_i can move only together with an entity of type t_{i-1}. The allowance constraint ensures that no two or more entities of the same type can occur in the same vertex at the same time. There is no special requirement on the initial and the goal arrangement λ_0 and λ_+ in the hierarchical subclass.

A task from Figure 1 belongs to the defined hierarchical subclass – mobile robots correspond to type t_1 while boxes correspond to type t_2. It is also easy to see that for a given relocation instance we can check in polynomial time if it is hierarchical or not. Observe that κ-hierarchical problem is also κ-bounded.

3.1. A Complete Solving Algorithm

The essential building block of a solving algorithm for the hierarchical sub-class is a procedure for solving cooperative path-finding problem. We may use polynomial time procedures for cooperative path-finding like *BIBOX* [5] (worst-case time complexity of $\mathcal{O}(|V|^3)$) or *Push-and-Swap* [3] (worst-case time complexity $\mathcal{O}(|V|^4)$). The path-finding procedure will be used to solve the problem on the individual levels of the hierarchy. As the entities on higher levels of hierarchy cannot move by themselves we always need to prepare supporting entities on the lower level to carry out each individual move of the solution provided by cooperative path-finding algorithm. The pseudo-code of the whole process is shown as Algorithm 1.

The algorithm first arranges entities of type t_κ that are on the top of the hierarchy and proceeds with $t_{\kappa-1}$ and so on until base level of type t_1 is reached – this process is represented by function *Solve-Hierarchical-Relocation*.

On the level h of the hierarchy we distinguish if it is a base level ($h = 1$) or not. In case of the base level, the problem already reduced to the cooperative path-finding and the path-finding algorithm can be called directly (lines 4-6 of *Solve-Hierarchical-Level*). If this is not the case, we produce the solution for the level h again by cooperative path-finding algorithm but now the moves cannot be performed directly but have to be supported by entities on lower levels (lines 8-12 of *Solve-Hierarchical-Level*). Supporting moves on lower hierarchy levels are implemented by recursive procedures *Move-Entity-on-Level* and *Prepare-for-Move-on-Level*. Each move of the entity on the level h requires that an entity on level $h - 1$ moves with it. Hence we need to ensure that there is some entity of type t_{h-1} in the source vertex and no entity of type t_{h-1} in the destination vertex. This requirement propagates to lower levels of the hierarchy recursively.

It is not difficult to observe that the algorithm is correct. It is just needed to verify that paths required on lines 3 and 12 of *Prepare-for-Move-on-Level* always exists which comes from the assumption of connectivity of G. Observe also that the algorithm is complete if the underlying cooperative path-finding procedure is complete which is true for both *BIBOX* and *Push-and-Swap*.

Proposition 1. Let $\kappa \in \mathbb{N}$ be a fixed height of the hierarchy. Then the κ-hierarchical relocation problem can be solved in the worst-case time of $\mathcal{O}(|V|^{\kappa+3})$ and the length of generated solution is $\mathcal{O}(|V|^{\kappa+2})$. ■

Proof. Let $T(h)$ denotes the time necessary to execute the procedure *Move-Entity-on-Level* on the level h. Then it holds that $T(1) = 1$ and $T(h) \leq |V| \cdot T(h-1) + |V|^2$ for $h > 1$ from which we quickly get that $T(h) = \mathcal{O}(|V|^h)$. In other words, one move on the level h consumes time of $\mathcal{O}(|V|^h)$. Similarly we can estimate the number of moves necessary to make a move on the level h. Let $M(h)$ denotes the number of moves to produced by *Move-Entity-on-Level* on the level h.

Then we have $M(1) = 1$ and $M(h) \leq |V| \cdot M(h-1)$ for $h > 1$ from which we obtain that $M(h) = \mathcal{O}(|V|^{h-1})$.

If we use the *BIBOX* algorithm (Surynek, 2009) for cooperative path-finding sub-problem which worst-case time complexity is $\mathcal{O}(|V|^3)$ and it produces solution consisting of $\mathcal{O}(|V|^3)$ moves then the worst-case time complexity for the last level of the hierarchy will be $\mathcal{O}(|V|^{\kappa+3})$ and it will produce $\mathcal{O}(|V|^{\kappa+2})$ moves. Accounting lower levels just adds the factor of κ which does not change the asymptotic estimations. ■

Algorithm 1. Solving algorithm for the κ-hierarchical relocation problem. A sequence of moves of entities is produced.

```
function Solve-Hierarchical-Relocation (Π): solution
1:  let $\Pi = [G = (V,E), Y, T = \{t_1, t_2, \dots, t_\kappa\}, \tau, \alpha, \beta, \lambda_0, \lambda_+]$
2:  $\lambda \leftarrow \lambda_0$; $\vec{s} \leftarrow []$
3:  for $h = \kappa, \kappa-1, \dots, 1$ do
4:      Solve-Hierarchical-Level ($[G, Y|_{t_1,\dots,t_h}, \{t_1, t_2, \dots, t_h\},$
            $\tau, \alpha, \beta, \lambda|_{t_1,t_2,\dots,t_h}, \lambda_+|_{t_1,t_2,\dots,t_h}], h$)
5:  return $\vec{s}$

procedure Solve-Hierarchical-Level ($\Pi, h$)
1:  let $\Pi = [G = (V,E), Y, T, \tau, \alpha, \beta, \lambda_0, \lambda_+]$
2:  if $h = 1$ then
3:      if $\lambda_0 \neq \lambda_+$ then
4:          $\vec{\sigma} \leftarrow$ Solve-Path-Finding ($G = (V,E), Y, \lambda_0, \lambda_+$)
5:          $\vec{s} \leftarrow \vec{s}.\vec{\sigma}$
6:          Make-Moves ($\lambda, \vec{\sigma}$)
7:  else
8:      if $\lambda_0|_{t_h} \neq \lambda_+|_{t_h}$ then
9:          $\vec{\sigma} \leftarrow$Solve-Path-Finding ($G = (V,E), Y|_{t_h}, \lambda_0|_{t_h}, \lambda_+|_{t_h}$)
10:         let $\vec{\sigma} = [y^1{:}\, u_1 \to v_1, y^2{:}\, u_2 \to v_2, \dots, y^m{:}\, u_m \to v_m]$
11:         for $i = 1,2,\dots,m$ do
12:             Move-Entity-on-Level ($u_i, v_i, h$)

procedure Move-Entity-on-Level ($u, v, h$)
    /* assumed $\lambda(u)|_{t_h} \neq \emptyset$ and $\lambda(v)|_{t_h} = \emptyset$ */
1:  if $h > 1$ then
2:      Prepare-for-Move-on-Level ($u, v, h-1$)
3:  $\vec{s} \leftarrow \vec{s}.[\lambda^{-1}(u){:}\, u \to v]$
4:  Make-Move ($\lambda, \lambda^{-1}(u){:}\, u \to v$)

procedure Prepare-for-Move-on-Level ($u, v, h$)
1:  if $\lambda(u)|_{t_h} = \emptyset$ then
2:      if $\lambda(v)|_{t_h} = \emptyset$ then
3:          let $[u = w_1, w_2, \dots, w_l]$ be a shortest path
                in $G$ to a vertex $w_l$ such that
                $\lambda(w_l)|_{t_h} \neq \emptyset$ and $\lambda(w_i)|_{t_h} = \emptyset$ for $i = 1,2,\dots,l-1$
4:          for $i = l, l-1, \dots, 2$ do
5:              Move-Entity-on-Level ($w_i, w_{i-1}, h$)
6:      else /* $\lambda(v)|_{t_h} \neq \emptyset$ */
7:          Move-Entity-on-Level ($v, u, h$)
8:  else /* $\lambda(u)|_{t_h} = \emptyset$ */
9:      if $\lambda(v)|_{t_h} = \emptyset$ then
10:         Move-Entity-on-Level ($u, v, h$)
11:     else /* $\lambda(v)|_{t_h} \neq \emptyset$ */
12:         let $[v = w_1, w_2, \dots, w_l]$ be a shortest path
                in $G$ to a vertex $w_l$ such that
                $\lambda(w_l)|_{t_h} = \emptyset$ and $\lambda(w_i)|_{t_h} \neq \emptyset$ for $i = 1,2,\dots,l-1$
13:         for $i = l, l-1, \dots, 2$ do
14:             Move-Entity-on-Level ($w_{i-1}, w_i, h$)
```

4. Conclusions and Discussion

A general concept of relocation problems has been introduced. It generalizes cooperative path-finding by imposing more constraints on allowed moved. An important hierarchical subclass of relocation problem has been also described. The hierarchical subclass is practically important as it allows to model planning problems like container transportation by a group of mobile robots. It has been shown that properly integrated algorithms for standard cooperative path-finding can used to solve the hierarchical subclass.

It will be interesting for future work to implement the presented algorithm for the hierarchical subclass and to evaluate its performance experimentally in comparison with domain-independent planners.

Acknowledgements

This work is supported by the Czech Science Foundation (contract number GAP103/10/1287) and by the Japan Society for the Promotion of Science (contract number P11743).

References

[1] Hearn, R. A., Demaine, E. D. "*PSPACE-completeness of sliding-block puzzles and other problems through the nondeterministic constraint logic model of computation",* Theoretical Computer Science, Volume 343(1-2), pp. 72-96, Elsevier, 2005.

[2] Helmert, M. "*Complexity Results for Standard Benchmark Domains in Planning",* Artificial Intelligence, Volume 143 (2), pp. 219-262, Elsevier Science Publishers, 2003.

[3] Luna, R., Berkis, K., E. "*Push-and-Swap: Fast Cooperative Path-Finding with Completeness Guarantees",* Proceedings of IJCAI, IJCAI Conference, 2011.

[4] Ryan, M. R. K. "*Exploiting subgraph structure in multi-robot path planning",* JAIR, Volume 31, pp. 497-542, AAAI Press, 2008.

[5] Surynek, P. "*An Application of Pebble Motion on Graphs to Abstract Multi-robot Path Planning",* Proceedings of ICTAI, pp. 151-158, IEEE Press, 2009.

[6] Wang. K. C., Botea, A. *Tractable Multi-Agent Path Planning on Grid Maps*. Proceedings of IJCAI, pp. 1870-1875, IJCAI Conference, 2009.