Object Reachability via Swaps along a Line

Sen Huang, Mingyu Xiao
School of Computer Science and Engineering
University of Electronic Science and Technology of China
{huangsen47, myxiao}@gmail.com

Abstract
The Housing Market problem is a widely studied resources allocation problem. In this problem, each agent can only receive a single object and has preferences over all objects. Starting from an initial endowment, we want to reach a certain assignment via a sequence of rational trades. We consider the problem whether an object is reachable for a given agent under a social network, where a trade between two agents is allowed if they are neighbors in the network and no participant has a deficit from the trade. Assume that the preferences of the agents are strict (no tie is allowed). This problem is polynomially solvable in a star-network and NP-complete in a tree-network. It is left as a challenging open problem whether the problem is polynomially solvable when the network is a path. We answer this open problem positively by giving a polynomial-time algorithm. Furthermore, we show that the problem on a path will become NP-hard when the preferences of the agents are weak (ties are allowed).

Introduction
Allocating indivisible objects to agents is an important problem in both computer science and economics. A widely studied setting is that each agent can only receive one single object and each agent has preferences over objects. This problem was previously called Assignment problem (Gardensdorfs 1973; Wilson 1977) and now we prefer to call it House Allocation problem (Abdulkadiroğlu and Sönmez 1998; Manlove 2013). When each agent is initially endowed with an object and we want to reallocate objects under some conditions without any monetary transfers, the problem is known as Housing Market problem (Shapley and Scarf 1974). Housing Market has several real-life applications such as allocation of housings (Abdulkadiroğlu and Sönmez 1999), organ exchange (Roth, Sönmez, and Ünver 2004) and so on. There are two different preference sets for agents. One is strict, which is a full ordinal list of all objects, and the other one is weak, where agents are allowed to be indifferent between objects. Both preference sets have been widely studied. Under strict preferences, the celebrated Top Trading Cycle rule (Shapley and Scarf 1974) has several key desirable properties. Some modifications of Top Trading Cycle rule, called Top Trading Absorbing Sets rule and Top Cycles rule, were introduced for weak preferences (Alcalde-Unzu and Molis 2011; Jaramillo and Manjunath 2012), which also hold some good properties. More studies of Housing Market under the two preference sets from different aspects can be found in the literature (Jaramillo and Manjunath 2012; Aziz and De Keijzer 2012; Saban and Sethuraman 2013; Ehlers 2014; Sonoda et al. 2014; Ahmad 2017).

Some rules allow a single exchange involving many agents. It is natural and fundamental to consider exchanges being bilateral deals (swaps), i.e., each exchange of objects happens only between two agents. A swap between two agents is allowed when they are mutually beneficial from the exchange. This natural rule for Housing Market has been studied in the literature (Damamme et al. 2015; Gourvès, Lesca, and Wilczynski 2017).

In some models, it is implicitly assumed that all agents have a tie with others. However, some agents often do not know each other and do not have the capacity to exchange even they can mutually get benefits. So Gourvès, Lesca, and Wilczynski (2017) studied Housing Market where the agents are embedded in a social network to denote the ability to exchange objects between them. In fact, recently it is a hot topic to study resources allocation problems over social networks and analyze the influences of networks. Abebe, Kleinberg, and Parkes (2017) and Bei, Qiao, and Zhang (2017) introduced social network of agents into the Fair Division problem of cake cutting. Bredereck, Kaczmarczyk, and Niedermeier (2018) and Beynier et al. (2018) also considered network-based Fair Division in allocating indivisible resources.

In this paper, we study Housing Market in a social network with simple trades between pairs of neighbors in the network. In this model, there are the same number of agents and objects and each agent is initially endowed with a single object. Each agent has preferences over all objects. The agents are embedded in a social network which determines
their ability to exchange their objects. Two agents may swap their items under two conditions: they are neighbors in the social network, and they find it mutually profitable (or no one will become worse under weak preferences). We focus on OBJECT REACHABILITY under this model: to determine whether an object is reachable for a given agent from the initial endowment via swaps. Damamme et al. (2015) firstly proved that the problem is NP-hard even to decide whether any one of a subset of objects is reachable for each agent. Gourvès, Lesca, and Wilczynski (2017) further showed that OBJECT REACHABILITY is polynomially solvable under star-structures and NP-complete under tree-structures. For the network being a path, they solved the special case where the given agent is an endpoint (a leaf) in the path and left it unsolved for the general case. All the above results are under strict preferences. In this paper, we will answer this open problem positively by giving a polynomial-time algorithm for OBJECT REACHABILITY in a path under strict preferences, and also prove that OBJECT REACHABILITY in a path under weak preferences is NP-hard.

Although paths are rather simple graph structures, OBJECT REACHABILITY in a path is not easy at all, as mentioned in (Gourvès, Lesca, and Wilczynski 2017) that “Despite its apparent simplicity, REACHABLE OBJECT (OBJECT REACHABILITY) in a path is a challenging open problem when no restriction on the agent’s location is made. We believe that this case is at the frontier of tractability.” Our algorithm involves several techniques and needs to call solvers for the 2-SAT problem, but it is still interesting.

The following part of the paper is organized as follows. Section 2 provides some backgrounds. Section 3 tackles the reachability of an object for an agent in a path-network under strict preferences. Section 4 shows the NP-hardness of OBJECT REACHABILITY in a path under weak preferences. The proofs of the lemmas and theorems marked with “*” are omitted due to the limitation of space, which can be found in the full version of this paper.

**Background**

**Model.** There are a set \( N = \{1, \ldots, n\} \) of \( n \) agents and a set \( O = \{o_1, \ldots, o_t\} \) of \( t \) objects. An assignment \( \sigma \) is a mapping from \( N \) to \( O \), where \( \sigma(i) \) is the object held by agent \( i \) in \( \sigma \). We also use \( \sigma^T(\sigma_i) \) to denote the agent who holds object \( o_i \) in \( \sigma \). Each agent holds exactly one object at all time. Initially, the agents are endowed with objects, and the initial endowment is denoted by \( \sigma_0 \). We assume w.l.o.g that \( \sigma_0(i) = o_i \) for every agent \( i \).

Each agent has preferences regarding objects. A strict preference is expressed as a full linear order of all objects. Agent \( i \)'s preference is denoted by \( >_i \), and \( o_i >_i o_0 \) indicates the fact that agent \( i \) prefers object \( o_i \) than object \( o_0 \). The whole strict preference profile for all agents is represented by \( > \). For weak preferences, agents are allowed to be indifferent between objects. For two disjoint subsets of objects \( S_1 \) and \( S_2 \), we use \( S_1 >_i S_2 \) to indicate that all objects in \( S_1 \) (resp., \( S_2 \)) are indifferent for agent \( i \) and agent \( i \) prefer any object in \( S_1 \) than any object in \( S_2 \). We use \( o_i \geq_0 o_0 \) to denote that agent \( i \) likes \( o_i \) at least as same as \( o_0 \). Two relations \( o_i \geq_0 o_0 \) and \( o_i \geq_1 o_0 \) together imply that \( o_i \) and \( o_0 \) are indifferent for agent \( i \). We may use \( \succeq \) to denote the whole weak preference profile for all agents.

Let \( G = (N, E) \) be an undirected graph as the social network among agents, where the edges capture the capability of communication and exchange between two agents. An instance of HOUSING MARKET is a tuple \( (N, O, >, G, \sigma_0) \) or \( (N, O, \succeq, G, \sigma_0) \) according to the preferences being strict or weak.

**Dynamics.** The approach we take in this paper is dynamic, and we focus on individually rational trades between two agents. A trade is individual rational if each participant receives an object at least as good as the one currently held, i.e., for two agents \( i \) and \( j \) and an assignment \( \sigma \), the trade between \( i \) and \( j \) on \( \sigma \) is individual rational if \( \sigma(j) \succeq_i \sigma(i) \) and \( \sigma(i) \succeq_j \sigma(j) \).

We require that every trade is performed between neighbors in the social network \( G \). Individual rational trades defined according to \( G \) are called swaps. A swap is an exchange, where two participants have the capability to communicate.

A sequence of swaps can be represented as a sequence of assignments \( (\sigma_0, \sigma_1, \sigma_2, \ldots, \sigma_t) \) such that for any \( i \in \{1, \ldots, t\} \), \( \sigma_i \) results from a swap from \( \sigma_{i-1} \). An assignment \( \sigma' \) is reachable if there exists a sequence of swaps \( (\sigma_0, \sigma_1, \sigma_2, \ldots, \sigma_t) \) such that \( \sigma_t = \sigma' \). An object \( o \in O \) is reachable for an agent \( i \in N \) if there is a sequence of swaps \( (\sigma_0, \sigma_1, \sigma_2, \ldots, \sigma_t) \) such that \( \sigma_t(i) = o \).

**Problems.** We consider the problem of checking whether an object is reachable for an agent from the initial endowment via swaps.

**OBJECT REACHABILITY**

**Instance:** \( (N, O, >, G, \sigma_0) \), an agent \( k \in N \), and an object \( o_k \in O \).

**Question:** Whether is object \( o_k \) reachable for agent \( k \)?
within the preferences. After a swap between agents 1 and 2 from \( \sigma_0 \) we get \( \sigma_1 \) and after a swap between agents 2 and 3 from \( \sigma_1 \) we get \( \sigma_2 \). The object \( o_1 \) is reachable for agent 3.

**Strict Object Reachability in a Path**

Strict Object Reachability is known to be NP-complete when the network is a tree and polynomially solvable when the network is a star (Gourvès, Lesca, and Wilczynski 2017). It is left unsolved whether the problem with the network being a path is NP-hard or not. We reveal some properties of Strict Object Reachability under the path structure and design a polynomial time algorithm for it. In the remaining part of this section, we assume that the preferences are strict and the network is a path.

Recall that the problem is to check whether an object \( o_k \) is reachable for an agent \( k \) with \( l < k \). The main idea of our algorithm is as follows. First, we show that the instance is equivalent to the instance after deleting all agents (and the corresponding endowed objects) on the left of agent \( l \).

Thus, we can assume the problem is to check whether object \( o_1 \) is reachable for an agent \( k \). Second, we prove that if \( o_1 \) is reachable for agent \( k \) then there is an object \( o_{n^t} \) with \( n^t \geq k \) that should be moved to agent \( k - 1 \) in the final assignment and we can ignore all agents and objects on the right of agent \( n^t \). We guess \( n^t \) by letting it be each possible value between \( k \) and \( n \) and get at most \( n \) candidate instances.

These instances are called neat \((o_1,o_{n^t},k)\)-Constrained instances. A neat \((o_1,o_{n^t},k)\)-Constrained instance contains only \( n^t \) agents and it is to check whether there is a reachable assignment \( \sigma^t \), called compatible assignment, such that \( \sigma^t(k) = o_1 \) and \( \sigma^t(k-1) = o_{n^t} \). Third, we are going to find compatible assignments. In a neat \((o_1,o_{n^t},k)\)-Constrained instance, in every compatible assignment each object \( o_l \) will be moved to either the left or the right of its original position in the path. We prove that for each direction, there is at most one possible position \( i_l \) (or \( i_r \) for the right direction) for each object \( o_l \). We can compute \( i_l \) and \( i_r \) directly in polynomial time. Since there are still two possible final positions for each object, we do not get a feasible assignment yet. Fourth, we reduce the subproblem to 2-SAT and determine which of \( i_l \) and \( i_r \) should be chose for each agent \( i \) by solving a 2-SAT instance. We show that each feasible assignment obtained in this step is corresponding to a reachable assignment for the neat \((o_1,o_{n^t},k)\)-Constrained instance.

Finally, we can solve the original problem in polynomial time, because the original instance is a yes-instance if and only if at least one of the candidate neat \((o_1,o_{n^t},k)\)-Constrained instances is a yes-instance.

In fact, when the preferences are strict, we have the following observations and lemmas.

**Observation 1.** Given a sequence of swaps \( \sigma_0, \sigma_1, \ldots, \sigma_t \) and an agent \( j \in N \). For any \( i \in \{0,1,\ldots,t-1\} \), it holds either \( \sigma_{i+1}(j) = \sigma_i(j) \) or \( \sigma_{i+1}(j) \succ \sigma_{i}(j) \).

It implies the following lemma.

**Lemma 1.** Given a sequence of swaps \( \sigma_0, \sigma_1, \ldots, \sigma_t \). For any two integers \( i < j \in \{0,1,\ldots,t\} \) and any agent \( q \in N \), if \( \sigma_i(q) = \sigma_j(q) \), then \( \sigma_d(q) = \sigma_i(q) \) for any integer \( i \leq d \leq j \).

Lemma 1 also says that an agent will not ‘visit’ an agent twice. This property is widely used in similar allocation problems under strict preferences.

Next, we analyze properties under the constraint that the social network is a path. In a swap, the moving of an object is on the right direction if it is moved from agent \( i \) to agent \( i+1 \), and the moving of an object is on the left direction if it is moved from agent \( i \) to agent \( i-1 \). In each swap, one object is moved on the right direction and one object is moved on the left direction. We study the tracks of the objects in a feasible assignment sequence.

**Lemma 2.** For a sequence of swaps \( \sigma_0, \sigma_1, \ldots, \sigma_t \), if \( \sigma_i^t(a_i) = j \) for an object \( o_i \), then there are exactly \( |j-i| \) swaps includes \( o_i \). Furthermore, all the \( |j-i| \) movings of \( o_i \) are on the right direction if \( i < j \), and all the \( |j-i| \) movings of \( o_i \) are on the left direction if \( i > j \).

**Lemma 3.** Let \((\sigma_0, \sigma_1, \ldots, \sigma_t)\) be a sequence of swaps, and \( a_0 \) and \( a_1 \) be two objects with \( a < b \). Let \( a' = \sigma_i^t(a_0) \) and \( b' = \sigma_i^t(a_0) \). If \( a' \preceq a \) or \( b' \succeq b \), then \( a' < b' \).

Lemma 1 shows that any object can only move on one direction. Lemma 3 shows that when an agent moves on the right direction, all objects initially allocated on the left of it can not move to the right of it at any time; when an agent moves on the left direction, all objects initially allocated on the right of it can not move to the left of it at any time.

In fact, if we want to move an object \( o_t \) to an agent \( k \) with \( k > l \), we may not need to move any object on the left of \( o_t \), i.e., objects \( o_{l'} \) with \( l' < l \). Equipped with Lemma 3, we can prove

**Lemma 4.** If object \( o_1 \) is reachable for agent \( k \), then there is a feasible assignment sequence \( (\sigma_0, \sigma_1, \ldots, \sigma_t) \) such that \( \sigma^t_1(a_1) = k \), and \( \sigma_i(l) = \sigma_i(0) \) for all \( i < l \) if \( l \leq k \) and for all \( i > l \) if \( l \geq k \).

For an instance \( I = (N, O, P, \sigma_0, k, o_1) \) of Strict Object Reachability in a path with \( l < k \), let \( I' = (N', O', P', \sigma_0', k, o_1) \) denote the instance obtained from \( I \) by deleting agents \( \{1,2,\ldots,l-1\} \) and objects \( \{o_1, o_2, \ldots, o_{l-1}\} \). In other words, we let \( N' = \{l, l+1, \ldots, n\}, O' = \{o_1, o_{l+1}, \ldots, o_n\}, \) and \( P', \sigma_0' \) be the corresponding subsets of \( P \) and \( \sigma_0 \).

**Lemma 5.** Object \( o_1 \) is reachable for agent \( k \) in the instance \( I \) if and only if object \( o_1 \) is reachable for agent \( k \) in the instance \( I' \).

By Lemma 5, we can always assume that the instance of Strict Object Reachability in a path is to check whether the object \( o_1 \) is reachable for an agent \( k \).

Assume that object \( o_1 \) is reachable for agent \( k \). For a sequence of swaps \( (\sigma_0, \sigma_1, \ldots, \sigma_t) \) such that \( \sigma_1(o_1) = k \), there are exactly \( k-1 \) swaps including \( o_1 \) which are moving \( o_1 \) on the right direction according to Lemma 2. The last swap including \( o_1 \) will be happened between agent \( k-1 \) and agent \( k \). Let \( o_{n'} \) denote the other object included in the last swap. In other words, after the last swap, agent \( k-1 \) will get the object \( o_{n'} \), and agent \( k \) will get the object \( o_1 \). Note that the moving of \( o_{n'} \), in this swap, is in the left direction. By Lemma 2, we know that all movings of \( o_{n'} \) in the sequence of swaps
are in the left direction. Therefore, we have the following observation.

**Observation 2.** It holds that \( n' \geq k \).

Our idea is to transform our problem to the following constrained problem: to determine whether there is a reachable assignment \( \sigma \) such that \( \sigma(k - 1) = o_n \) and \( \sigma(k) = o_1 \), where \( n' \geq k \). We do not know the exact value of \( n' \). So we search by letting \( n' \) be each value in \( \{ k, k + 1, \ldots, n \} \). This will only increase the running time bound by a factor of \( n \). We denote the above constrained problem by \((o_1, o_{n'}, k)\)-CONSTRAINED problem.

**Lemma 6.** An instance \( I = (N, O, \succ, P, \sigma_0, k, o_1) \) is yes if and only if at least one of the \((o_1, o_{n'}, k)\)-CONSTRAINED instances for \( n' \in \{k, k + 1, \ldots, n\} \) is yes.

For an \((o_1, o_{n'}, k)\)-CONSTRAINED instance \( I \), we use \( I_{-n'} \) to denote the instance obtained from \( I \) by deleting agents \( \{n' + 1, n' + 2, \ldots, n\} \) and objects \( \{o_{n' + 1}, o_{n' + 2}, \ldots, o_n\} \).

**Lemma 7.** An \((o_1, o_{n'}, k)\)-CONSTRAINED instance \( I \) is yes if and only if the instance \( I_{-n'} \) is yes.

Let \((o_1, o_{n'}, k)\)-CONSTRAINED instance be a compatible assignment for a neat \((o_1, o_{n'}, k)\)-CONSTRAINED instance only. For any two integers \( a \) and \( b \), we use \([a, b]\) to denote the set of integers between \( a \) and \( b \) (including \( a \) and \( b \)).

**Lemma 8.** Let \((\sigma_0, \sigma_1, \ldots, \sigma_t)\) be a sequence of swaps, and \( o_a \) and \( o_b \) be two objects with \( a < b \). Let \( a' = \sigma_1^T(o_a) \) and \( b' = \sigma_1^T(o_b) \). Assume that \( Q = [a, a'] \cap [b, b'] \). If \( a' > a \) and \( b' > b \), it holds that \( o_a \succ_q o_b \) for all \( q \in Q \).

See Figure 1 for an illustration for Lemma 8(a).

![Figure 1: An illustration for Lemma 8(a)](image)

**Lemma 9.** Let \((\sigma_0, \sigma_1, \ldots, \sigma_t)\) be a sequence of swaps for a neat \((o_1, o_{n'}, k)\)-CONSTRAINED instance such that \( \sigma_1^T(o_1) = k \) and \( \sigma_1^T(o_{n'}) = k - 1 \). Let \( Q' = [a + 1, a' + 1] \cap [b, b'] \). If \( Q' \neq \emptyset \), there is a swap including \( o_a \) and \( o_b \) which happens between agent \( c - 1 \) and \( c \), where \( c = a' + b' - k + 1 \) in \( Q' \).

See Figure 2 for an illustration for Lemma 9.

![Figure 2: An illustration for Lemma 9](image)

Given a neat \((o_1, o_{n'}, k)\)-CONSTRAINED instance and an assignment \( \sigma \) such that \( \sigma_1^T(o_1) = k \) and \( \sigma_1^T(o_{n'}) = k - 1 \). We show some conditions for \( \sigma \) to be a feasible assignment. For any two objects \( o_a \) and \( o_b \), we let \( a' = \sigma_1^T(o_a) \) and \( b' = \sigma_1^T(o_b) \). We say \( o_a \) and \( o_b \) are intersected if \( Q = [a, a'] \cap [b, b'] \) is not an empty set. There are three kinds of intersections, which are corresponding to Lemma 8(a), Lemma 8(b) and Lemma 9. We say a pair of objects \( o_a \) and \( o_b \) are compatible in assignment \( \sigma \) if there are either not intersected or intersected and satisfying one of the follows:

1. when \( a' > a \) and \( b' > b \), it holds that \( a' < b' \) (corresponding to Lemma 10).
2. when \( a' < a \) and \( b' < b \), it holds that \( a' < b' \) (corresponding to Lemma 10).
3. when \( a' > a \) and \( b' < b \), it holds that \( c = a' + b' - k + 1 \) in \( Q' = Q \setminus \{a\} \), \( o_a \succ_q o_b \) for all \( q \in Q' \), \( o_a \succ_q o_b \) for all \( c \leq q \leq \min(a', b') \) (corresponding to Lemma 9).

An assignment \( \sigma \) is compatible if it holds \( \sigma(i) \neq o_i \) for any agent \( i \) and any pair of objects in it are compatible.

**Lemma 10.** If \( \sigma_1 \) is a reachable assignment for a neat \((o_1, o_{n'}, k)\)-CONSTRAINED instance such that \( \sigma_1^T(o_1) = k \) and \( \sigma_1^T(o_{n'}) = k - 1 \), then \( \sigma_1 \) is compatible.

**Lemma 11.** Let \( \sigma_1 \) be a compatible assignment for a neat \((o_1, o_{n'}, k)\)-CONSTRAINED instance such that \( \sigma_1^T(o_1) = k \) and \( \sigma_1^T(o_{n'}) = k - 1 \). For any two objects \( o_{x-1} \) and \( o_x \), if \( \sigma_1^T(o_{x-1}) > x - 1 \) and \( \sigma_1^T(o_x) < x \), then the swap between agents \( x - 1 \) and \( x \) in \( o_0 \) is feasible. Let \( \sigma_1 \) denote the assignment after the swap between \( x - 1 \) and \( x \) in \( o_0 \). Then \( \sigma_1 \) is still compatible by taking \( \sigma_1 \) as the initial endowment.

Based on Lemma 11 we will prove the following lemma.

**Lemma 12.** Let \( \sigma_1 \) be an assignment for a neat \((o_1, o_{n'}, k)\)-CONSTRAINED instance such that \( \sigma_1^T(o_1) = k \) and \( \sigma_1^T(o_{n'}) = k - 1 \). If \( \sigma_1 \) is compatible, then \( \sigma_1 \) is a reachable assignment.

By Lemma 12, to solve a neat \((o_1, o_{n'}, k)\)-CONSTRAINED instance, we only need to find a compatible assignment.

---

1If the swap is happened between agents 1 and 2, then object \( o_1 \) will be moved to agent 2 in \( \sigma_1 \) and we will may be not able to get an \((o_1, o_{n'}, k)\)-CONSTRAINED instance by letting \( \sigma_1 \) be the initial endowment. For this case, we will delete agent 1 and object \( \sigma_1(1) \) to keep object \( o_1 \) in the first agent. More details will be given in the full version.
Computing Compatible Assignments

In a compatible assignment, object $o_i$ will be assigned to agent $k$ and object $o_{j'}$ will be assigned to agent $k-1$. We consider other objects $o_i$ for $i \in \{2, 3, \ldots, n' - 1\}$. In a compatible assignment, object $o_i$ will not be assigned to agent $i$ since each agent will attend in at least one swap including object $o_i$ or $o_{j'}$. There are two possible cases: $o_i$ is assigned to agent $i'$ such that $i' < i$; $o_i$ is assigned to agent $i'$ such that $i' > i$. We say that $o_i$ is moved to the left side for the former case and moved to the right side for the latter case. We will show that for each direction, there is only one possible position for each object $o_i$ in a compatible assignment.

First, we consider $i \in \{2, 3, \ldots, k - 1\}$. Assume that object $o_i$ is moved to the left side in a compatible assignment. Thus, $o_i$ and $o_{j'}$ are intersected and the intersection is of the case in Lemma 9. We find the index $i'$ such that $i' \leq i$, $o_i \succ_{i'-1} o_i$ and $o_i \succ_{j'} o_i$ for each $j \in \{i', i'+1, \ldots, i\}$. We can see that $i'$ is the only possible agent for object $o_i$ to make $o_i$ and $o_{j'}$ compatible if $o_i$ is moved to the left side. We use $i'$ to denote this agent if it exists for $i$. Assume that object $o_i$ is moved to the right side in a compatible assignment. Since $o_i$ will be moved to agent $k$ and $o_{j'}$ will be moved to the right side, by Lemma 3 we know that $o_i$ will be moved to the right of $o_i$, i.e., an agent $i''$ with $i'' > k$. Thus, $o_i$ and $o_{i''}$ are intersected, and the intersection is of the case in Lemma 9. We find the index $i'$ such that $i' > k$, $o_i \succ_{i''} o_{i''}$ and $o_{i''} \succ_j o_i$ for each $j \in \{k - 1, k, \ldots, i' - 1\}$. We can see that $i'$ is the only possible agent for object $o_i$ to make $o_{i''}$ and $o_i$ compatible if $o_i$ is moved to the right side. We use $i'$ to denote this agent if it exists for $i$.

Second, we consider $i \in \{k, k + 1, \ldots, n' - 1\}$. In fact, the structure of neat $(o_i, o_{i''}, k)$-constrained instances is symmetrical. We can rename the agents on the path from left to right as $\{n', n' - 1, \ldots, 1\}$ instead of $\{1, 2, \ldots, n'\}$ and then this case becomes the above case. We can compute $i_1$ and $i_2$ for each $i \in \{k, k + 1, \ldots, n' - 1\}$ in a similar way. Therefore, we can compute $i_1$ and $i_2$ for all $i \in \{2, 3, \ldots, n' - 1\}$ if they exist.

If none of $i_1$ and $i_2$ exists for some $i$, then this instance is a no-instance. If only one of $i_1$ and $i_2$ exists, then object $o_i$ must be assigned to this agent in any compatible assignment. The hardest case is that both of $i_1$ and $i_2$ exist, where we may not know which agent the object will be assigned to in the compatible assignment. For this case, we will rely on algorithms for 2-SAT to find possible solutions.

For each agent $j$, we will use $R_j$ to store all possible objects that may be assigned to agent $j$ in a compatible assignment. We use the following procedure to maintain $R_j$. Initially, we let $R_{k-1} = \{o_{j'}\}$, $R_k = \{o_i\}$, and $R_i = \emptyset$ for all other agent $i$. Then for each $i \in \{2, 3, \ldots, n' - 1\}$, we compute $i_1$ and $i_2$, and add $o_i$ into $R_{i_1}$ and $R_{i_2}$. After this, we can do the following to make the size of each $R_j$ at most 2. While there is a set $R_{j_0}$ becoming an empty set, stop and report the instance is a no-instance; while there is a set $R_{j_0}$ containing only one object $o_{j_0}$, and the object $o_{j_0}$ appears in two sets $R_{j_0}$ and $R_{j_0}'$, delete $o_{j_0}$ from $R_{j_0}'$. The correctness of the third step is based on the fact that agent $j_0$ should get one object. If there is only one candidate object $o_{j_0}$ for agent $j_0$, then $o_{j_0}$ can only be assigned to agent $j_0$, no possible to agent $j_0'$.

We also analyze the running time of the above procedure to compute $R_j$. For each object $o_i$, we can compute $i_1$ and $i_2$ in $O(n)$. Therefore, we use $O(n^2)$ time to set the values for all sets $R_j$ in the first two steps. To update $R_i$, we may execute at most $n$ iterations in the third step and each iteration can be executed in $O(n)$. Therefore, the procedure running times in $O(n^2)$ time.

Lemma 13.* After the above procedure, either the instance is a no-instance or it holds that $1 \leq |R_j| \leq 2$ for each $j \in \{1, 2, \ldots, n\}$.

For a set $R_j$ containing only one object $o_i$, we know that the object $o_i$ should be assigned to agent $j$ in any compatible assignment. For sets $R_j$ containing two objects, we still need to design which object is assigned to this agent such that we can get a compatible assignment.

We will reduce the remaining problem to 2-SAT. The instance contains $n'$ variables $\{x_1, x_2, \ldots, x_{n'}\}$ corresponding to the $n'$ objects. When $x_i = 1$, it means that the object $o_i$ moves on the right direction, i.e., we will assign it to the agent $i_r$ in the compatible assignment. When $x_i = 0$, it means that the object $o_i$ moves on the left direction and we will assign it to the agent $i_l$ in the compatible assignment.

We have two kinds of clauses, called agent clauses and compatible clauses.

For each set $R_j$, we associate $|R_j|$ literals with it. If there is an object $o_i$ such that $i = j$, we associate literal $\overline{x_i}$ with $R_j$; if there is an object $o_i$ such that $i = j$, we associate literal $x_i$ with $R_j$. For each set $R_j$ of size 1 (let the associated literal be $\ell_j$), we construct one clause containing only one literal $c_j : \ell_j$. For each set $R_j$ of size 2 (let the associated literals be $\ell_j^1$ and $\ell_j^2$), we construct two clauses $c_{j1} : \ell_j^1 \lor \overline{\ell_j^2}$ and $c_{j2} : \overline{\ell_j^1} \lor \ell_j^2$. These clauses are called agent clauses. The agent clauses are to guarantee that exactly one object is assigned to each agent.

For each pair of sets $R_j$ and $R_i$, we construct several clauses according to the definition of compatibility. For any two objects $o_j \in R_j$ (the corresponding literal associated to $R_j$ is $\ell_j$) and $o_i \in R_i$ (the corresponding literal associated to $R_i$ is $\ell_i$), we say that $\ell_j$ and $\ell_i$ are compatible if $o_j$ and $o_i$ are compatible when $o_j$ is assigned to agent $j$ and $o_i$ is assigned to agent $i$ in the assignment. If $\ell_j$ and $\ell_i$ are not compatible, then either $o_j$ cannot be assigned to agent $j$ or $o_i$ cannot be assigned to agent $i$ in any compatible assignment. So we construct one compatible clause: $\overline{\ell_j} \lor \overline{\ell_i}$ for each pair of incompatible pair $\ell_j$ and $\ell_i$. Since each set contains at most two objects, for each pair of sets $R_j$ and $R_i$, we will create at most $2 \times 2 = 4$ clauses. In fact, when there are four clauses for a pair, the instance will become a no-instance, since no matter what objects assigned to agents $j$ and $i$, there is no compatible assignment. In the following example, we will illustrate the construct of agent clauses and compatible clauses.

We can see that each clause contains at most two literals and then the instance is a 2-SAT instance. The construction of 2-SAT instances implies that

Lemma 14. The 2-SAT instance has a feasible assign-
ment if and only if the corresponding neat \((o_1, o_{n'}, k)\)-CONSTRAINED instance has a compatible assignment.

The main steps of the whole algorithm to solve **Strict Object Reachability** in a path are listed in Algorithm 1. The correctness of the algorithm follows from Lemma 5, Lemma 6, Lemma 7 and Lemma 14. Next, we analyze the running time bound of it. The dominating part of the running time is taken by the computation for neat \((o_1, o_{n'}, k)\)-CONSTRAINED instances. Next, we consider the running time used for solving each neat \((o_1, o_{n'}, k)\)-CONSTRAINED instance. By the above analysis, we use \(O(n^2)\) time to compute the final values for all sets \(R_j\). To construct 2-SAT instance, we construct at most \(2n\) agent clauses in \(O(n)\) time and construct at most \(4n)\) compatible clauses, each of which will take \(O(n)\) time to check the compatibility. So the 2-SAT instance can be constructed in \(O(n^3)\) time. We use the \(O(n+m)\)-time algorithm for 2-SAT (Aspvall, Plass, and Tarjan 1979) to solve our instance, where \(m = O(n^2)\). There are at most \(n\) neat \((o_1, o_{n'}, k)\)-CONSTRAINED instances. In total, we use \(O(n^4)\) time.

**Theorem 1.** **Strict Object Reachability** in a path can be solved in \(O(n^4)\) time.

**Algorithm 1:** Main steps to solve **Strict Object Reachability** in a path

**Input:** An instance \((N, O, \succ, P, \sigma, k, n \in N, o_i \in O)\)

**Output:** To determine whether \(o_1\) is reachable for \(k\)

for \(k \leq n' \leq n\) do

1. Construct the neat \((o_1, o_{n'}, k)\)-CONSTRAINED instance by deleting agent \(i\) and object \(o_i\) for all \(n' < i \leq n\);
2. Compute \(i_r\) and \(i_l\) for all \(1 \leq i \leq n'\) if it exist according to Lemma 9;
3. Construct the set \(R_j\) of all possible objects that may be assigned to each agent \(j\) according to the result in the above step, where \(R_{k-1} = \{o_{n'}\}\) and \(R_k = \{o_1\}\);
4. Iteratively update \(R_j\) according to the procedure before Lemma 13;
5. Construct a 2-SAT instance as follows: construct a variable for each object, construct agent clauses according to \(R_j\), and construct compatible clauses for all incompatible pairs;
6. Determine whether the 2-SAT instance is satisfiable;
7. if the 2-SAT instance is yes then
8. return yes;
9. return no.

We give an example to show the steps to compute a compatible assignment for a neat \((o_1, o_{n'}, k)\)-CONSTRAINED instance.

**Example 2.** Consider a neat \((o_1, o_{n'}, k)\)-CONSTRAINED instance with \(n' = 8\) and \(k = 5\) as the top right figure. We compute \(i_r\) and \(i_l\) for all \(1 \leq i \leq n'\) by the above procedure, the values of which are listed in the top right table.

<table>
<thead>
<tr>
<th>agent (i)</th>
<th>(i_l)</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
</tr>
</thead>
<tbody>
<tr>
<td>(i_r)</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

We construct \(R_j\) for each agent \(j\) according to this table, and then update them as doing in the procedure. After the update, it holds that \(1 \leq |R_j| \leq 2\) for all \(1 \leq j \leq n'\). We reduce the remaining problem to 2-SAT. The agent clauses for each set \(R_j\) are given in the last column of the table.

<table>
<thead>
<tr>
<th>Set</th>
<th>Initial</th>
<th>Updated</th>
<th>Agent clauses</th>
</tr>
</thead>
<tbody>
<tr>
<td>(R_1)</td>
<td>(o_2)</td>
<td>(x_2)</td>
<td>(x_1)</td>
</tr>
<tr>
<td>(R_2)</td>
<td>(o_3, o_5)</td>
<td>(o_3, o_5)</td>
<td>(x_3 \lor x_5, x_3 \lor x_5)</td>
</tr>
<tr>
<td>(R_3)</td>
<td>(o_1, o_6)</td>
<td>(o_1, o_6)</td>
<td>(x_4 \lor x_6, x_4 \lor x_6)</td>
</tr>
<tr>
<td>(R_4)</td>
<td>(o_8)</td>
<td>(o_8)</td>
<td>(x_7)</td>
</tr>
<tr>
<td>(R_5)</td>
<td>(o_1)</td>
<td>(o_1)</td>
<td>(x_1)</td>
</tr>
<tr>
<td>(R_6)</td>
<td>(o_2, o_3, o_5)</td>
<td>(o_3, o_5)</td>
<td>(x_3 \lor x_5, x_3 \lor x_5)</td>
</tr>
<tr>
<td>(R_7)</td>
<td>(o_4, o_6)</td>
<td>(o_4, o_6)</td>
<td>(x_4 \lor x_6, x_4 \lor x_6)</td>
</tr>
<tr>
<td>(R_8)</td>
<td>(o_7)</td>
<td>(o_7)</td>
<td>(x_7)</td>
</tr>
</tbody>
</table>

Next, we construct compatible clauses for all incompatible pairs. We check all pairs of objects and find that there are only two incompatible cases: \(o_4\) and \(o_5\) are incompatible if \(o_4\) and \(o_5\) are moved to agent 3 and agent 2, respectively; \(o_4\) and \(o_5\) are incompatible if \(o_4\) and \(o_5\) are moved to agent 7 and agent 6, respectively. The compatible clauses are

\[ x_4 \lor x_5 \lor x_3 \lor x_3. \]

By using the \(O(n+m)\) time algorithm for 2-SAT (Aspvall, Plass, and Tarjan 1979), we get a feasible variables assignment \((1, 0, 0, 0, 1, 1, 1, 0)\) for the 2-SAT instance. The corresponding swap sequence constructed via variables assignment above is given below.

**Remark:** Although Step 4 of our algorithm will compute possible values \(i_l\) and \(i_r\) for each \(i\), it does not mean that object \(o_i\) must be reachable for \(i_l\) or \(i_r\). In the above example, object \(o_2\) is not reachable for agent \(2_r = 6\) since agent 3 prefers its initial object \(o_3\) to \(o_2\) and then \(o_2\) cannot go to the right direction. In our algorithm, the compatible clauses can avoid assigning an object to an unreachable value \(i_l\) or \(i_r\). In the above example, if object \(o_2\) goes to agent 6, then some object \(o_i\) with \(i > 2\) will go to agent 1 or 2 and we will get an incompatible pair \(o_2\) and \(o_i\).
In the initial assignment $\sigma_0$, object $t$ is assigned to agent $T$, object $c_i$ is assigned to agent $C_i$ for $i \in \{1, 2, \ldots, m\}$, object $q_i$ is assigned to agent $A_i^q$ for $i \in \{1, 2, \ldots, n\}$ and $j \in \{1, 2, 3\}$, and $\sigma_i^{\text{ini}}$ (resp., $\sigma_i^{\text{ini}}$ and $\sigma_i^{\text{ini}}$) is assigned to agent $X_i^{\text{ini}}$ (resp., $X_i^{\text{ini}}$ and $X_i^{\text{ini}}$) for each $i \in \{1, 2, \ldots, n\}$.

Next, we define the preference profile $\succeq$. We only show the objects that each agent prefers at least as its initial one and all other objects can be put behind its initial endowment in any order. The initial endowment is denoted by a square in the preference. Let $L_i$, be the set of the objects associated with the literals in clause $C_i$, i.e., $L_i$ is the set of objects $\sigma_i^{\text{ini}}$, $\sigma_i^{\text{ini}}$ and $\sigma_i^{\text{ini}}$ with $n_a = i$, $p_b = i$ or $q_c = i$. For each variable $v_i$, we define a set of objects $W_i = \{c_1, \ldots, c_m\} \cup \{\sigma_i^{\text{ini}} : j > i\} \cup \{\sigma_i^{\text{ini}} : j \neq i\} \cup \{\sigma_i^{\text{ini}} : j > i\} \cup \{\sigma_{i+1}^{\text{ini}} : j < i, l = 1, 2, 3\}$. We are ready to give the preferences for the agents.

First, we consider the preferences for $T$ and $C_i$. The following preferences ensure that when $C_i$ holds an object in $L_i$ for each $i \in \{1, \ldots, m\}$, object $t$ is reachable for agent $C_m$ via a sequence of $m$ swaps between $C_i$ and $C_{i-1}$ for $i = 1, \ldots, m$, where $C_0 = T$.

$T: L_1 \succ \{t\}$
$C_i: L_{i+1} \succ t \succ L_i \succ c_1 \succ L_{i-1} \succ \ldots \succ c_{i-1} \succ L_1 \succ c_i$, for all $1 \leq i \leq m - 1$
$C_m: t \succ L_m \succ c_1 \succ L_{m-1} \succ \ldots \succ c_{m-1} \succ L_1 \succ c_m$.

Next, we consider the preferences for the agents in each block $B_i$. The following preferences ensure that at most one of $\sigma_i^{\text{ini}}$ and $\{\sigma_i^{\text{ini}}, \sigma_i^{\text{ini}}\}$ can be moved to the right of the block, which will indicate the corresponding variable is either true or false. If $\sigma_i^{\text{ini}}$ is moved to the right of the block, we will assign the corresponding variable false; if some of $\{\sigma_i^{\text{ini}}, \sigma_i^{\text{ini}}\}$ is moved to the right of the block, we will assign the corresponding variable true. We use the preference of $A_i^3$ to control this. Furthermore, we use $A_i^1$, $A_i^2$ and $A_i^3$ to (temporarily) hold $\sigma_i^{\text{ini}}$ (or $\sigma_i^{\text{ini}}$ and $\sigma_i^{\text{ini}}$) if they do not need to be moved to the right of the block.

The instance $I_{\text{WOR}}$ contains $6n + m + 1$ agents and objects, which are constructed as follows. There is an agent named $T$. For each clause $C_i (i \in \{1, \ldots, m\})$ we introduce an agent also named $C_i$. For each variable $v_i$, we add six agents, named as $X_i^{\text{ini}}$, $X_i^{\text{ini}}$, $X_i^{\text{ini}}$, $A_i^2$, $A_i^3$ and $A_i^1$. They form a path of length 5 in the order as showed below. We call the path a block and denote it by $B_i$. The names of the six agents have certain meaning: $X_i^{\text{ini}}$ means that the negative literal of $v_i$ appears in the clause $C_{ni}$; $X_i^{\text{ini}}$ and $X_i^{\text{ini}}$ mean that the positive literals of $v_i$ appear in the two clauses $C_{pi}$ and $C_{qi}$; $A_i^3$, $A_i^2$ and $A_i^1$ are three auxiliary agents.

\[
\begin{array}{cccccc}
X_i^{\text{ini}} & X_i^{\text{ini}} & X_i^{\text{ini}} & A_i^2 & A_i^3 & A_i^1 \\
B_m & B_1 & C_m & \cdots & C_1 & T
\end{array}
\]

The whole path is connected in the order showed below.

\[
\begin{array}{cccccc}
P : 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \\
\sigma_0 : o_1 & o_2 & o_3 & o_4 & o_5 & o_6 & o_7 & o_8 \\
\sigma_1 : o_2 & o_1 & o_3 & o_4 & o_5 & o_6 & o_7 & o_8 \\
\sigma_2 : o_2 & o_3 & o_1 & o_4 & o_5 & o_6 & o_7 & o_8 \\
\sigma_3 : o_2 & o_3 & o_4 & o_1 & o_5 & o_6 & o_7 & o_8 \\
\sigma_4 : o_2 & o_3 & o_4 & o_1 & o_5 & o_6 & o_7 & o_8 \\
\sigma_5 : o_2 & o_3 & o_4 & o_1 & o_5 & o_6 & o_7 & o_8 \\
\sigma_6 : o_2 & o_3 & o_4 & o_1 & o_5 & o_6 & o_7 & o_8 \\
\sigma_7 : o_2 & o_3 & o_4 & o_1 & o_5 & o_6 & o_7 & o_8 \\
\end{array}
\]
literal corresponding to the object \( a \in L_{i+1} \) to be true for all agents \( C_i \) to get a satisfying assignment for \( I_{SAT} \) because the construction of each block \( B_i \) does not allow both \( \overline{v}_i \) and one of \( o_i^p \) and \( o_i^q \) moving to the right of the block according to the construction of the block \( B_i \) the preference of \( A_i^3 \).

On the other hand, if there is a satisfied assignment \( \tau \) for \( I_{SAT} \), we can construct a reachable assignment for \( I_{WOR} \). For each variable \( v_i \), if it is true in \( \tau \), we move \( o_i^p \) and \( o_i^q \) to agents \( A_i^3 \) and \( A_i^2 \); if it is false in \( \tau \), we move \( \overline{v}_i \) to agent \( A_i^1 \). These objects are called true objects. All these happen within each block. After this procedure, we move true objects \( \overline{v}_i \), \( o_i^p \) and \( o_i^q \) to agent \( C_j \) with \( j = n_a \), \( j = p_b \) or \( j = q \), one by one in an order where \( C_j \) with smaller \( j \) first gets its object in \( L_j \). During this procedure, once another true object is moved out of its position \( A_i^3 \) or \( A_i^2 \) (this may happen when the true object is on the moving path of another true object to \( C_j \)), we will simply move it back by one swap. So in each iteration only one true object is moved out of its current position \( A_i^3 \) or \( A_i^2 \) and it is moved to its final position \( C_j \) directly.

\[ \Box \]

Conclusion

In this paper, we mainly investigate Object Reachability that asks whether an object is reachable for a given agent. We show that when the network is a path, Weak Object Reachability is NP-hard but Strict Object Reachability is polynomially solvable. In the literature, more problems under the Housing Market model with a different objective have been investigated, such as the reachability of a whole assignment, finding Pareto efficient assignments and so on (Gourvès, Lesca, and Wilczynski 2017). Some of our results can be extended to these problems with some modifications. We will give the details in the full version of this paper.

References


