Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence (AAAI-16)

Multi-Agent Path Finding with Payload Transfers and the Package-Exchange Robot-Routing Problem*

Hang Ma
Department of Computer Science
University of Southern California
hangma@usc.edu

Craig Tovey
School of Industrial and Systems Engineering
Georgia Institute of Technology
ctovey@isye.gatech.edu

Guni Sharon
Department of Information Systems Engineering
Ben-Gurion University of the Negev
gunisharon@gmail.com

T. K. Satish Kumar and **Sven Koenig**
Department of Computer Science
University of Southern California
tkskwork@gmail.com and skoenig@usc.edu

Abstract

We study transportation problems where robots have to deliver packages and can transfer the packages among each other. Specifically, we study the package-exchange robot-routing problem (PERR), where each robot carries one package, any two robots in adjacent locations can exchange their packages, and each package needs to be delivered to a given destination. We prove that exchange operations make all PERR instances solvable. Yet, we also show that PERR is NP-hard to approximate within any factor less than $4/3$ for makespan minimization and is NP-hard to solve for flowtime minimization, even when there are only two types of packages. Our proof techniques also generate new insights into other transportation problems, for example, into the hardness of approximating optimal solutions to the standard multi-agent path-finding problem (MAPF). Finally, we present optimal and suboptimal PERR solvers that are inspired by MAPF solvers, namely a flow-based ILP formulation and an adaptation of conflict-based search. Our empirical results demonstrate that these solvers scale well and that PERR instances often have smaller makespans and flowtimes than the corresponding MAPF instances.

Introduction

Payloads can be transferred in real-world applications such as ride-sharing (or taxis) with passenger transfers (Coltin and Veloso 2014) or package delivery with robots in offices (Veloso et al. 2015). The theoretical implications of allowing payload transfers are still poorly understood. In this paper, we therefore study the package-exchange robot-routing problem (PERR), where each robot carries one package, any two robots in adjacent locations can exchange their packages, and each package needs to be delivered to a given destination. We want to minimize the time when the last package reaches its destination (makespan) or the sum of the times when each package reaches its destination (flowtime). PERR is motivated by the observation that the robots are often interchangeable but the packages are not - due to their different destinations.

PERR is identical to the standard multi-agent path-finding problem (MAPF), except that MAPF does not permit exchange operations: Each robot needs to move to a given destination but two robots in adjacent locations cannot exchange their positions. MAPF is NP-hard to solve for both makespan and flowtime minimization (Yu and LaValle 2013b). Furthermore, several optimal and suboptimal MAPF solvers have been developed, including (Kornhauser, Miller, and Spirakis 1984; Silver 2005; Sturtevant and Buro 2006; Wagner and Choset 2011; Surynek 2012; Erdem et al. 2013; de Wilde, ter Mors, and Witteveen 2013; Sharon et al. 2013; Goldenberg et al. 2014; Cohen, Uras, and Koenig 2015).

Figures 1(a) & 1(b) illustrate that exchange operations can improve the makespan and flowtime by arbitrary factors. We prove by construction that exchange operations make all PERR instances solvable, and derive polynomial upper bounds on makespan and flowtime. Yet, we also show that PERR is NP-hard to approximate within any factor less than $4/3$ for makespan minimization and is NP-hard to solve for flowtime minimization, even when there are only two types of packages (where packages of the same type are interchangeable). Our proof techniques also generate new insights into other transportation problems, for example, into the hardness of approximating optimal MAPF solutions.

Finally, we present optimal and suboptimal PERR solvers that are inspired by MAPF solvers, namely a flow-based ILP formulation (Yu and LaValle 2013a) and an adaptation of conflict-based search (Sharon et al. 2015). Our empirical results demonstrate that these solvers scale well and that PERR instances often have smaller makespans and flowtimes than the corresponding MAPF instances.

*We thank Jingjin Yu for making the code of their flow-based MAPF solver and Nathan Sturtevant for making game maps available to us. The research at USC was supported by NSF under grant numbers 1409987 and 1319966. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the sponsoring organizations, agencies or the U.S. government.

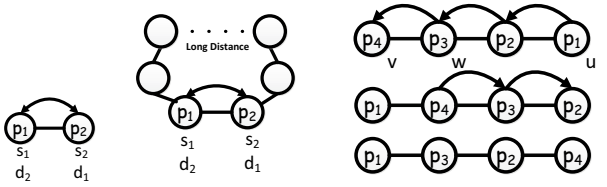

Figure 1: (a) (b) Motivating examples that demonstrate the power of exchange operations. Two robots carrying packages p_1 and p_2 have to deliver them to their destination vertices d_1 and d_2, respectively. They can exchange their packages. If exchange operations are not allowed in (a), then no solution exists. If exchange operations are not allowed in (b), then one robot must take the long path. (c) Package p_1 can be moved from vertex u to vertex v via a series of exchange operations along the path $\langle u \dots v \rangle$. Then, package p_4 can be moved from vertex w to u via a series of exchange operation along the path $\langle w \dots u \rangle$, restoring the original positions of packages p_2 and p_3.

Package-Exchange Robot-Routing Problem

We formalize the package-exchange robot-routing problem (PERR) as follows: We are given an undirected connected graph $G = (V, E)$ and M packages $\{p_1, p_2 \dots p_M\}$. Each package p_i has a source vertex s_i and a destination vertex d_i. Let $l_i(t)$ be the vertex of p_i at time $t = 0 \dots \infty$. A *plan* assigns a function l_i to each package p_i. A solution is a plan that satisfies the following conditions: 1. For all packages i, $l_i(0) = s_i$ (each package starts at its source vertex). 2. For all packages i, there exists a time T_i^{end} such that, for all times $t = T_i^{end} \dots \infty$, $l_i(t) = d_i$ (each package ends at its destination vertex). We assume without loss of generality that T_i^{end} is the smallest such time. 3. For all packages i and all times t, $(l_i(t), l_i(t+1)) \in E$ or $l_i(t) = l_i(t+1)$ (each package repeatedly either moves to an adjacent vertex in a single time step or does not move). 4. For all pairs of different packages i and j and all times t, $l_i(t) \neq l_j(t)$ (two packages are not at the same vertex at the same time). The *makespan* of a solution is $\max_i T_i^{end}$, and the *flowtime* of the solution is $\sum_i T_i^{end}$.

We also study a generalization of PERR, namely the K-type package-exchange robot-routing problem (K-PERR), where the packages and destination vertices are partitioned into K types. If there are M_k packages of type k, then there must also be M_k destination vertices of type k. Packages of the same type are interchangeable: Each package of type k must be delivered to a different destination vertex of type k. PERR is a special case of K-PERR with $K = M$.

In PERR and K-PERR, two packages p_i and p_j can be exchanged in a single time step by the robots carrying them when they are at adjacent vertices. If this happens, then p_i and p_j traverse the same edge in opposite directions, resulting in $l_i(t) = l_j(t+1)$ and $l_j(t) = l_i(t+1)$ at the corresponding time t.

Feasibility

Figure 1(a) shows that some instances of multi-agent path finding problems (MAPF) are not solvable. The following theorem proves that all PERR instances are solvable:

Theorem 1. *All PERR instances are solvable. Solutions with polynomial makespans and flowtimes can be found in polynomial time.*

Proof. Without loss of generality, we assume that all vertices are initially occupied by robots carrying packages. (In the following, if neither of two adjacent vertices is occupied by a robot, then an exchange operation involving them is replaced by a no-op operation. If only one of them is occupied by a robot, then the exchange operation is replaced by the robot moving to the adjacent vertex.) Then, any two packages can switch vertices without affecting the vertices of the other packages. To see this, consider two packages p_i and p_j and their current vertices u and v, respectively. Let $\langle u \dots w, v \rangle$ be a shortest path from u to v, where w is the vertex on the path directly before v. A series of exchange operations along this path moves p_i to v and every other package on this path against the path one edge closer to u (in at most $|V| - 1$ time steps). In particular, it moves p_j to w. A series of exchange operations against the path $\langle u \dots w \rangle$ then moves p_j to u and every other package on this path back to its original vertex (in at most $|V| - 2$ time steps), hence proving the claim. Figure 1(c) shows an example. This property allows one to route all packages to their destination vertices one at a time. Our algorithm performs only a polynomial number of operations, which implies that the makespans and flowtimes of the resulting solutions are also polynomial. □

Letting any two packages switch vertices, as done in the proof of Theorem 1, takes at most $(|V| - 1) + (|V| - 2) = 2|V| - 3$ time steps. This operation needs to be repeated at most $M - 1$ times since one additional package reaches its destination vertex each time. An upper bound on the makespan of the resulting solutions is thus

$$U_m = (M - 1)(2|V| - 3). \tag{1}$$

Since each package reaches its destination vertex by time U_m, an upper bound on the flowtime of the resulting solutions is

$$U_f = MU_m = M(M - 1)(2|V| - 3). \tag{2}$$

The proof of Theorem 1 applies unchanged to K-PERR if all packages of the same type are first assigned arbitrary different destination vertices of the same type, yielding the following corollary:

Corollary 2. *All K-PERR instances are solvable. Solutions with polynomial makespans and flowtimes can be found in polynomial time.*

Intractability

We now prove that PERR is NP-hard to approximate within any factor less than $4/3$ for makespan minimization and

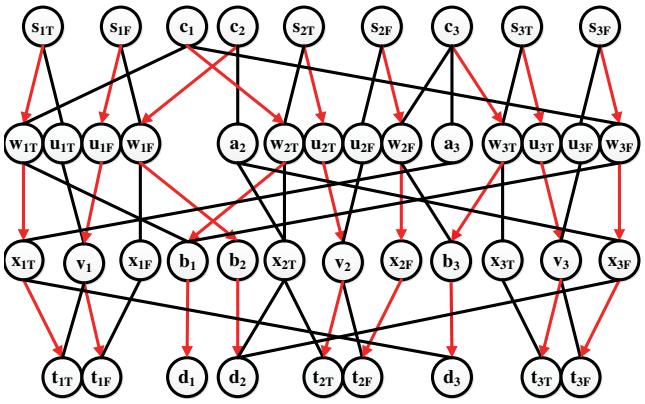

Figure 2: A PERR instance reduced from the ≤3,=3-SAT instance $(X_1 \vee X_2 \vee \overline{X}_3) \wedge (\overline{X}_1 \vee X_2 \vee \overline{X}_3) \wedge (X_1 \vee \overline{X}_2 \vee X_3)$. Clause C_1 is the first clause that literal X_1 appears in. The corresponding clause path is $\langle c_1, w_{1T}, b_1, d_1 \rangle$. Since clause C_3 is the second clause that X_1 appears in, vertex a_3 is introduced. The corresponding clause path is $\langle c_3, a_3, x_{1T}, d_3 \rangle$. The red (directed) edges represent one optimal solution to the PERR instance of makespan three, which corresponds to the satisfying assignment $(X_1, X_2, X_3) =$ (*False*,*True*,*True*).

is NP-hard to solve for flowtime minimization, by reducing an NP-complete version of the satisfiability problem, called ≤3,=3-SAT (Tovey 1984), to PERR. A ≤3,=3-SAT instance consists of n Boolean variables $X_1 \ldots X_n$ and m disjunctive clauses $C_1 \ldots C_m$. Each variable appears in exactly three clauses, uncomplemented at least once and complemented at least once. Each clause contains at most three literals. The decision question asks whether the instance is satisfiable.

Theorem 3. *For any $\epsilon > 0$, it is NP-hard to find a $4/3 - \epsilon$ approximate solution to PERR for makespan minimization.*

Proof. We construct a PERR instance that has a solution with makespan three if and only if a given ≤3,=3-SAT instance is satisfiable. Figure 2 shows an example.

For each variable X_i in the ≤3,=3-SAT instance, we construct two "literal" packages, p_{iT} and p_{iF}, with source vertices s_{iT} and s_{iF} and destination vertices t_{iT} and t_{iF}, respectively. For each literal package, we construct two paths to get to its destination vertex in three time steps: a "shared" path, namely $\langle s_{iT}, u_{iT}, v_i, t_{iT} \rangle$ for p_{iT} and $\langle s_{iF}, u_{iF}, v_i, t_{iF} \rangle$ for p_{iF}, and a "private" path, namely $\langle s_{iT}, w_{iT}, x_{iT}, t_{iT} \rangle$ for p_{iT} and $\langle s_{iF}, w_{iF}, x_{iF}, t_{iF} \rangle$ for p_{iF}. The shared paths for p_{iT} and p_{iF} intersect at vertex v_i. Only one of the two paths can thus be used if a makespan of three is to be achieved. Sending literal package p_{iT} (or p_{iF}) along the shared path corresponds to assigning *True* (or *False*) to X_i in the ≤3,=3-SAT instance.

For each clause C_j in the ≤3,=3-SAT instance, we construct a "clause" package p_j with source vertex c_j and destination vertex d_j. It has multiple (but at most three) "clause" paths to get to its destination vertex in three time steps, which have a one-to-one correspondence to the literals in C_j. Every literal X_i (or $\overline{X_i}$) can appear in at most two clauses. If C_j is the first clause that it appears in, then the clause path is $\langle c_j, w_{iT}, b_j, d_j \rangle$ (or $\langle c_j, w_{iF}, b_j, d_j \rangle$). If C_j is the second clause that it appears in, a vertex a_j is introduced and the clause path is instead $\langle c_j, a_j, x_{iT}, d_j \rangle$ (or $\langle c_j, a_j, x_{iF}, d_j \rangle$). The clause path of each C_j with respect to any literal in that clause and the private path of the literal intersect. Only one of the two paths can thus be used if a makespan of three is to be achieved.

Suppose that a satisfying assignment to the ≤3,=3-SAT instance exists. Then, a solution with makespan three is obtained by sending literal packages of true literals along their shared paths, the other literal packages along their private paths and clause packages along the clause paths corresponding to one of the true literals in those clauses.

Conversely, suppose that a solution with makespan three exists. Then, each clause package traverses the clause path corresponding to one of the literals in that clause, and the corresponding literal package traverses its shared path. Since the packages of a literal and its complement cannot both use their shared path if a makespan of three is to be achieved, we can assign *True* to every literal whose package uses its shared path without assigning *True* to both the uncomplemented and complemented literals. If the packages of both literals use their private paths, we can assign *True* to any one of the literals and *False* to the other one. A solution to the PERR instance with makespan three thus yields a satisfying assignment to the ≤3,=3-SAT instance.

To summarize, the PERR instance has a solution with makespan three if and only if the ≤3,=3-SAT instance is satisfiable. Also, the PERR instance cannot have a solution with makespan smaller than three and always has a solution with makespan four, even if the ≤3,=3-SAT instance is unsatisfiable. For any $\epsilon > 0$, any approximation algorithm for PERR with ratio $4/3 - \epsilon$ thus computes a solution with makespan three whenever the ≤3,=3-SAT instance is satisfiable and therefore solves ≤3,=3-SAT. □

The PERR instance in the proof of Theorem 3 has the property that the length of every path from a source vertex to the corresponding destination vertex is at least three. Thus, if the makespan is three, then every package is delivered in exactly three time steps and the flowtime is $3M$. Moreover, if the makespan exceeds three, then the flowtime exceeds $3M$, yielding the following corollary:

Corollary 4. *It is NP-hard to minimize flowtime for PERR.*

The construction in the proof of Theorem 3 almost applies in case all literal packages are of the same type and all clause packages are of the same type - but not quite since, if clause C_k is the first clause that literal X_i appears in and clause C_j is the second such clause, then clause package p_k could travel from its source vertex c_k along path $\langle c_k, w_{iT}, x_{iT}, d_j \rangle$ of length three to destination vertex d_j. Thus, a solution to the PERR instance with makespan three does not necessarily yield a satisfying assignment to the ≤3,=3-SAT instance. We therefore now prove that even 2-PERR is NP-hard to approximate within any factor less than $4/3$ for makespan minimization and is NP-hard to solve for flowtime minimization, by reducing a different NP-complete version of the sat-

isfiability problem, called $2/\overline{\mathbf{2}}/3$-SAT, to 2-PERR. A $2/\overline{\mathbf{2}}/3$-SAT instance consists of n Boolean variables $X_1 \dots X_n$ and m disjunctive clauses $C_1 \dots C_m$. Each variable appears complemented in one clause of size two, appears uncomplemented in one clause of size two and appears a third time in a clause of size three. The decision question asks whether the instance is satisfiable.

Lemma 5. *$2/\overline{2}/3$-SAT is NP-complete.*

Proof. $2/\overline{2}/3$-SAT is clearly in NP. 3-SAT is NP-complete and can be reduced to $2/\overline{2}/3$-SAT as follows, similar to (Tovey 1984): Given a standard 3-SAT instance with variables Y_i and exactly three literals per clause, we delete all clauses that contain a variable that does not appear in any other clause. Then, we consider each remaining variable Y_i in turn. Let $K_i > 1$ be the number of literals that it occurs in. We replace the k-th occurrence of variable Y_i by a new variable $X_{i,k}$, that is, replace literal Y_i (or $\overline{Y}_i$) with literal $X_{i,k}$ (or $\overline{X}_{i,k}$). Then, we append the following clauses of two literals each to the constructed instance: $\left(\bigwedge_{k=1}^{K_i-1}(X_{i,k} \vee \overline{X}_{i,k+1})\right) \wedge (X_{i,K_i} \vee \overline{X}_{i,1})$. The clause $X_{i,k} \vee \overline{X}_{i,k+1}$ implies that $X_{i,k+1}$ must be false if $X_{i,k}$ is false and that $X_{i,k}$ must be true if $X_{i,k+1}$ is true. The cyclic structure of the clauses therefore forces all $X_{i,1} \dots X_{i,K_i}$ to have the same truth value. Thus, the constructed instance is satisfiable if and only if the original 3-SAT instance is satisfiable. Each $X_{i,k}$ appears complemented in one clause of size two, appears uncomplemented in one clause of size two and appears a third time in a clause of size three. Moreover, the transformation requires only polynomial time. □

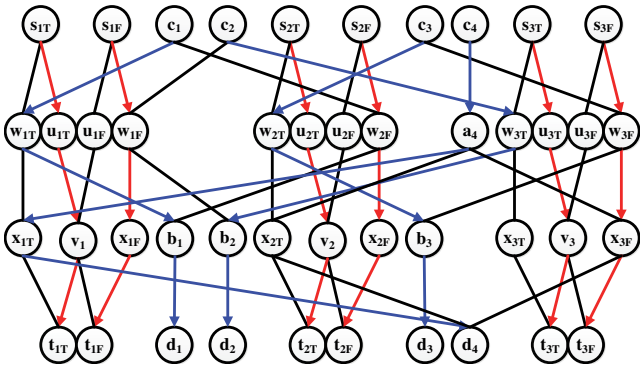

Figure 3: A 2-PERR instance reduced from the $2/\overline{2}/3$-SAT instance $(X_1 \vee \overline{X}_2) \wedge (\overline{X}_1 \vee X_3) \wedge (X_2 \vee \overline{X}_3) \wedge (X_1 \vee X_2 \vee \overline{X}_3)$. Consider any solution with makespan three. The only paths of length three from source vertex c_4 to a destination vertex of a clause package have the form $\langle c_4, a_4, x_{iT}/x_{iF}, d_4\rangle$. Clause package p_4 thus must arrive at destination vertex d_4. The only paths of length three from source vertex c_1 to a destination vertex of a clause package have the form $\langle c_1, w_{iT}/w_{iF}, b_1, d_1\rangle$ or $\langle c_1, w_{1T}, x_{1T}, d_4\rangle$. Paths of the latter form cannot be used because d_4 must receive p_4. Clause package p_1 thus must arrive at destination vertex d_1. The colored (directed) edges represent one optimal solution to the 2-PERR instance with makespan three, which yields the satisfying assignment $(X_1, X_2, X_3) =$ (*True*,*True*,*True*). The red edges represent paths of the literal packages, and the blue edges represent paths of the clause packages.

Theorem 6. *For any $\epsilon > 0$, it is NP-hard to find a $4/3-\epsilon$ approximate solution to 2-PERR for makespan minimization.*

Proof. We construct a 2-PERR instance that has a solution with makespan three if and only if a given $2/\overline{2}/3$-SAT instance is satisfiable. Figure 3 shows an example. Then, the remainder of the proof of Theorem 3 applies without change.

We follow the construction in the proof of Theorem 3, with the exception of making every clause path for clauses c_j of size two of the form $\langle c_j, w_{iT}/w_{iF}, b_j, d_j\rangle$ (vertex a_j is not introduced in this case) and every clause path for clauses c_k of size three of the form $\langle c_k, a_k, x_{iT}/x_{iF}, d_k\rangle$. We distinguish only two package types, namely literal and clause packages. Every vertex s_{iT} and s_{iF} (or t_{iT} and t_{iF}) is a source (or destination) vertex of a literal package, and every vertex c_j (or d_j) is a source (or destination) vertex of a clause package.

Suppose that a satisfying assignment to the $2/\overline{2}/3$-SAT instance exists. Then, a solution with makespan three is obtained, as in the proof of Theorem 3, by sending literal packages of true literals along their shared paths, the other literal packages along their private paths and clause packages along the clause paths corresponding to one of the true literals in those clauses.

Conversely, suppose that a solution with makespan three exists. Then, the only paths of length three from source vertex s_{iT} or s_{iF} to a destination vertex of a literal package are the shared or private paths that end at destination vertex t_{iT} or t_{iF}. The shared paths of packages p_{iT} and p_{iF} intersect at their penultimate vertices. Since the two packages cannot occupy the same vertex at time $t = 2$, at most one of them can traverse its shared path if a makespan of three is to be achieved. Package p_{iT} cannot arrive at t_{iF} since then p_{iF} has no path of length three to a destination vertex of a literal available, and vice versa for p_{iF}. Now consider an arbitrary clause C_k of size three. The only paths of length three from source vertex c_k to a destination vertex of a clause package have the form $\langle c_k, a_k, x_{iT}/x_{iF}, d_k\rangle$. Clause package p_k thus must arrive at destination vertex d_k. Finally, consider an arbitrary clause C_j of size two. The only paths of length three from source vertex c_j to a destination vertex of a clause package have the form $\langle c_j, w_{iT}/w_{iF}, b_j, d_j\rangle$ or $\langle c_j, w_{iT}/w_{iF}, x_{iT}/x_{iF}, d_{k'}\rangle$, where the clause $C_{k'}$ of size three shares a literal with clause C_j. Paths of the latter form cannot be used because destination vertex $d_{k'}$ must receive clause package $p_{k'}$. Clause package p_j thus must arrive at destination vertex d_j. The situation is now identical to that in the proof of Theorem 3 because every package p_{iT} (or p_{iF}, p_j or p_k) travels from its source vertex s_{iT} (or s_{iF}, c_j or c_k) to destination vertex t_{iT} (or t_{iF}, d_j or d_k). A solution to the 2-PERR instance with makespan three thus yields a satisfying assignment to the $2/\overline{2}/3$-SAT instance when assigning *True* to every literal whose package uses its shared

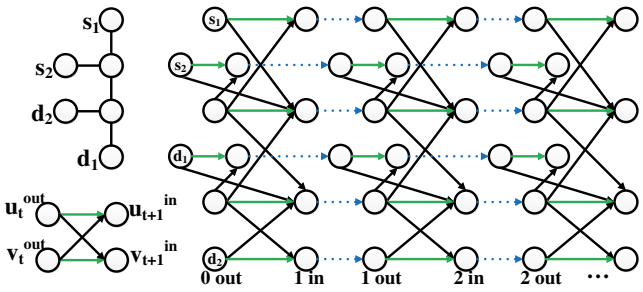

Figure 4: An example of reducing a PERR instance to an integer multi-commodity network-flow instance.

path, as explained in the proof of Theorem 3.

□

Corollary 7. *It is NP-hard to minimize flowtime for 2-PERR.*

Theorem 6 and Corollary 7 hold not only for 2-PERR but also for K-PERR for all $K = 3 \dots M$ because one can pad the graph of the constructed 2-PERR instance with additional vertices, each being both the source and destination of a package of a different type. This padding leaves the makespan and flowtime of any solution unchanged.

Generalizations

None of our proofs require exchange operations. Thus, they apply unchanged to many variants of PERR, including where 1. robots cannot exchange packages; 2. packages or robots disappear upon delivery; 3. robots can carry more than one package; and 4. robots exchange packages more slowly or more quickly than moving along an edge. In particular, our proofs apply unchanged to MAPF, even if there are only two types of robots (where each robot of a given type must move to a different destination vertex of the same type), yielding the following corollary:

Corollary 8. *For any $\epsilon > 0$, it is NP-hard to find a $4/3 - \epsilon$ approximate solution to MAPF for makespan minimization, even if there are only two types of robots.*

Corollary 8 improves the state-of-the-art NP-hardness result of MAPF for makespan minimization (Yu and LaValle 2013b), which is based on reducing $2/2/4$-SAT to the $(n^2 - 1)$-puzzle (Ratner and Warmuth 1990), since it shows not only the NP-hardness of solving MAPF but also the NP-hardness of approximating it with constant-factor approximations. Their proof does not transfer to PERR.

Network Flow and PERR

We now reduce K-PERR to the integer multi-commodity network-flow problem. We then use this reduction to solve PERR optimally.

Reducing K-PERR to Multicommodity Flow

Given a K-PERR instance and a fixed number of time steps T, we construct a directed flow network N with vertex set $V' = \bigcup_{v \in V} \bigcup_{t=0}^{T} \{v_t^{in}, v_t^{out}\}$. Vertex v_t^{in} (or v_t^{out}) represents vertex $v \in V$ in the beginning (or at the end) of time t. For each $1 \leq i \leq M$, set a supply of one at source vertex $(s_i)_0^{out}$ and a demand of one at destination vertex $(d_i)_T^{out}$, both for commodity type k where k is the type of package p_i. Construct edge set E' as follows: All edges are directed and have unit capacity. For all $v \in V$ and $t = 0 \dots T-1$, create (green) edges $(v_t^{out}, v_{t+1}^{in})$ to allow a robot to stay at the vertex. For all $(u, v) \in E$ and $t = 0 \dots T-1$, create (black) edges $(u_t^{out}, v_{t+1}^{in})$ and $(v_t^{out}, u_{t+1}^{in})$ to allow a robot to move along the edge or exchange a package along the edge. For all $v \in V$ and $t = 1 \dots T$, also create (blue dashed) edges (v_t^{in}, v_t^{out}) to prevent more than one robot from being at the same vertex at the same time. Figure 4 shows an example. The construction implies the following theorem:

Theorem 9. *Every feasible integer multi-commodity flow on $N = (V', E')$ yields a K-PERR solution with makespan of at most T, and vice versa.*

The proof of this theorem mirrors the one for the reduction of MAPF to the integer multi-commodity network-flow problem (Yu and LaValle 2013a).

Computing Optimal Solutions

We use our reduction to solve PERR for makespan and flowtime minimization. An integer multi-commodity network-flow problem can be expressed as an integer linear program (ILP) using the following standard formulation. Let $\delta^+(v)$ (or $\delta^-(v)$) be the set of incoming (or outgoing) edges of vertex v. The 0/1 variable $x_i[e]$ represents the amount of flow of type i on edge e.

$$\forall e \in E' \qquad 0 \leq \sum_{i=1}^{M} x_i[e] \leq 1$$

$$\forall i = 1 \dots M \quad \forall v \in V \setminus \{(s_i)_0^{out}, (d_i)_T^{out}\} \qquad \sum_{e \in \delta^+(v)} x_i[e] - \sum_{e \in \delta^-(v)} x_i[e] = 0.$$

$$\forall i = 1 \dots M \qquad \sum_{e \in \delta^-((s_i)_0^{out})} x_i[e] = \sum_{e \in \delta^+((d_i)_T^{out})} x_i[e] = 1$$

Minimizing Makespan Binary search on T finds the minimum value of T for which feasible solution exists, similar to (Yu and LaValle 2013a). This requires a lower and an upper bound on T. For the former we use the maximum over i of the length of a shortest path from s_i to d_i in G. Our upper bound is given by Equation 1. Thus, the binary search requires at most $O(\log U_m) = O(\log(M|V|)) \leq O(\log |V|^2) = O(\log |V|)$ iterations. In our experiments, we actually assign the lower bound to T and then repeatedly increment T until we reach feasibility (since we noticed that we get feasibility right away). However, we have developed a way to minimize makespan by solving one ILP only, namely by using our upper bound for T and adding a single auxiliary variable z. We minimize z subject to $z \geq (t+1)x_i[(u_t^{out}, v_{t+1}^{in})]$ for all $i = 1 \dots M$, $(u, v) \in E$ and $t = 0 \dots U-1$. The last movement of any package sets the value of z, thereby minimizing the makespan.

Minimizing Flowtime A similar formulation allows us to minimize flowtime by solving one ILP only, namely by using an upper bound for T and adding auxiliary variables z_i for all $i = 1 \dots M$. We minimize $\sum_{i=1}^{M} z_i$ subject to $z_i \geq (t+1)x_i[(u_t^{out}, v_{t+1}^{in})]$ for all $i = 1 \dots M$, $(u, v) \in E$ and $t = 0 \dots U - 1$. The last movement of package p_i sets the value of z_i, thereby minimizing the flowtime.

The Special Case of 1-PERR

In the special case of 1-PERR, the integer multi-commodity network-flow problem becomes a regular feasible circulation problem, which is easily converted to a maximum flow problem. Since all supply and demand values are one, any polynomial or pseudopolynomial time algorithm for maximum flow determines the feasibility of 1-PERR for any particular T in polynomial time. Binary search on T yields the following corollary:

Corollary 10. *Minimizing makespan for 1-PERR is solvable in polynomial time.*

In variant 2 of 1-PERR, where packages disappear upon delivery, flow time can be minimized in polynomial time by setting all edge costs to one and adding "disappearance" edges $((d_i)_t^{out}, (d_i)_T^{out})$ which for each destination vertex d_i permit a package there at any time t to disappear. This creates a minimum cost flow problem with unit costs, which therefore can be solved in polynomial time by any polynomial or pseudopolynomial time algorithm.

Experiments

We now report our experimental results for makespan minimization. We implemented a flow-based PERR solver based on a software package provided by the authors of (Yu and LaValle 2013a). Our flow-based solver casts a PERR instance as a series of integer multi-commodity network-flow problems as described in Section "Network Flow and PERR", each of which is formulated as an ILP and given to the ILP solver Gurobi 6.0. We also use a version of this flow-based solver that determines a solution only on the subgraph given by the set of shortest individual paths of all packages from their source vertices to their destination vertices. This version sacrifices optimality for reducing the size of the graph to be searched. Both flow-based solvers are written in Java. Additionally, we adapted the Conflict-Based Search (CBS) algorithm (Sharon et al. 2015) from MAPF to PERR, which requires only the addition of exchange operations. The adapted CBS solver is correct, complete and optimal, as can be shown with arguments similar to those in (Sharon et al. 2015) using the upper bound from Equation 1. It is written in C#. We ran all solvers on a 2.50GHz Intel Core i5-2450M computer with 6GB RAM, a 1.5GB JVM and a single thread. We performed three experiments.

In the first experiment, we studied the effect of exchange operations by solving both PERR instances and the corresponding MAPF instances (which do not permit exchange operations) on small benchmark maps with three to sixteen robots each (Sajid, Luna, and Bekris 2012). Table 1 reports the makespans of the MAPF solutions found by Sequential Push and Swap (Luna and Bekris 2011), Parallel Push and Swap (Sajid, Luna, and Bekris 2012), Sequential Push and Swap with Parallelization, ODA* (Standley 2010; Standley and Korf 2011) and CBS. The first three solvers are suboptimal ones, and the last two solvers are optimal ones. The results for the first four solvers are copied from (Sajid, Luna, and Bekris 2012). Table 1 also reports the makespans of the PERR solutions found by adapted CBS, the optimal flow-based solver and the suboptimal flow-based solver. The first two solvers are optimal ones, and the last solver is a suboptimal one. Both flow-based solvers managed to compute optimal solutions for all PERR instances within 0.1 seconds each. Adapted CBS computed optimal solutions for all PERR instances but reached the ten-minute runtime limit for the last one, while CBS solved even fewer MAPF instances. Table 1 suggests that the PERR instances are easier to solve than the corresponding MAPF instances and have smaller makespans.

Table 1: Makespans on small benchmark maps (with the numbers of exchange operations for some PERR solutions given in parentheses). Dashed entries indicate that the ten-minute runtime limit was reached.

Map	MAPF					PERR				
	Sequ. PS	Par. PS	Sequ. PS with Par.	ODA*	CBS	Adaptive CBS	Optimal Flow		Suboptimal Flow	
Tree	39	9	20	6	6	3	3	(2)	3	(2)
Corners	68	21	27	8	8	8	8	(2)	8	(2)
Tunnel	159	44	49	6	-	4	4	(6)	4	(6)
Connect	126	37	29	-	-	11	11	(9)	11	(9)
String	39	15	23	8	8	6	6	(4)	6	(4)
TwoLoop	5,269	1,015	-	-	-	-	8	(40)	8	(40)

In the second experiment, we compared the scalability of the three PERR solvers on four-neighbor 20×15 connected grid maps. We generated thirty instances with randomly blocked cells and random source and destination vertices each for different numbers of robots (varied from 10 to 50 in increments of 10) and obstacle densities (varied from 0% to 30% in increments of 5%). The two flow-based solvers solved all instances, while the adapted CBS solver solved only some of them. Table 2 and Figure 5 suggest that the flow-based solvers perform better on instances with many robots, while the adapted CBS solver performs better on instances with few robots.

In the third experiment, we ran the adapted CBS solver on the benchmark map brc202d from the video game Dragon Age: Origins (Sturtevant 2012), discretized to a grid map of 254,930 cells. We generated thirty instances with random source and destination vertices each for different numbers of robots (varied from 5 to 50 in increments of 5). Both flow-based solvers reached the ten-minute runtime limit on this sparse grid map with many bottlenecks. They do not scale well for ILPs with large numbers of time steps. Table 3 suggests that the adapted CBS solver scales well. It is thus possible to adapt a MAPF solver to PERR with little effort and have it perform well on a grid map from an actual game.

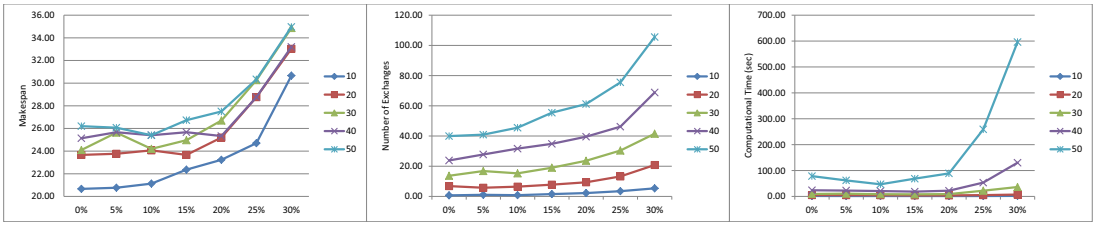

Figure 5: Makespans, numbers of exchange operations and runtimes of the optimal flow-based solver for different numbers of robots (colors) and obstacle densities (x-axis).

Conclusion

We studied the package-exchange robot-routing problem (PERR) as a first step toward understanding more general transportation problems with payload exchanges (and transfers). There is a continuum of problems. *One robot* yields the long-studied classic rural postman problem. *As many robots as packages* yields the less understood MAPF and PERR problems. Understanding the extremes, as done in this paper, improves our ability to attack the middle in future work, where many real-world applications are positioned.

References

Cohen, L.; Uras, T.; and Koenig, S. 2015. Feasibility study: Using highways for bounded-suboptimal multi-agent path finding. In *Annual Symposium on Combinatorial Search*, 2–8.

Coltin, B., and Veloso, M. 2014. Scheduling for transfers in pickup and delivery problems with very large neighborhood search. In *AAAI Conference on Artificial Intelligence*, 2250–2256.

de Wilde, B.; ter Mors, A.; and Witteveen, C. 2013. Push and rotate: Cooperative multi-agent path planning. In *International Conf. on Autonomous Agents and Multi-Agent Systems*, 87–94.

Erdem, E.; Kisa, D. G.; Öztok, U.; and Schüller, P. 2013. A general formal framework for pathfinding problems with multiple agents. In *AAAI Conference on Artificial Intelligence*.

Goldenberg, M.; Felner, A.; Stern, R.; Sharon, G.; Sturtevant, N.; Holte, R.; and Schaeffer, J. 2014. Enhanced partial expansion A*. *Journal of Artificial Intelligence Research* 50:141–187.

Kornhauser, D.; Miller, G.; and Spirakis, P. 1984. Coordinating pebble motion on graphs, the diameter of permutation groups, and applications. In *Annual Symposium on Foundations of Computer Science*, 241–250.

Luna, R., and Bekris, K. 2011. Push and swap: Fast cooperative path-finding with completeness guarantees. In *International Joint Conference on Artificial Intelligence*, 294–300.

Ratner, D., and Warmuth, M. 1990. The (n^2-1)-puzzle and related relocation problems. *J. of Symbolic Computation* 10(2):111 – 137.

Sajid, Q.; Luna, R.; and Bekris, K. 2012. Multi-agent pathfinding with simultaneous execution of single-agent primitives. In *Annual Symposium on Combinatorial Search*.

Sharon, G.; Stern, R.; Goldenberg, M.; and Felner, A. 2013. The increasing cost tree search for optimal multi-agent pathfinding. *Artificial Intelligence* 195:470–495.

Sharon, G.; Stern, R.; Felner, A.; and Sturtevant, N. 2015. Conflict-based search for optimal multi-agent pathfinding. *Artificial Intelligence* 219:40–66.

Silver, D. 2005. Cooperative pathfinding. In *Artificial Intelligence and Interactive Digital Entertainment*, 117–122.

Standley, T., and Korf, R. 2011. Complete algorithms for cooperative pathfinding problems. In *International Joint Conference on Artificial Intelligence*, 668–673.

Standley, T. 2010. Finding optimal solutions to cooperative pathfinding problems. In *AAAI Conference on Artificial Intelligence*.

Sturtevant, N., and Buro, M. 2006. Improving collaborative pathfinding using map abstraction. In *Artificial Intelligence and Interactive Digital Entertainment*, 80–85.

Sturtevant, N. 2012. Benchmarks for grid-based pathfinding. *Transactions on Computational Intelligence and AI in Games* 4(2):144 – 148.

Surynek, P. 2012. Towards optimal cooperative path planning in hard setups through satisfiability solving. In *Pacific Rim International Conference on Artificial Intelligence*, 564–576.

Tovey, C. 1984. A simplified NP-complete satisfiability problem. *Discrete Applied Mathematics* 8:85–90.

Veloso, M.; Biswas, J.; Coltin, B.; and Rosenthal, S. 2015. Cobots: Robust symbiotic autonomous mobile service robots. In *International Joint Conference on Artificial Intelligence*, 4423.

Wagner, G., and Choset, H. 2011. M*: A complete multirobot path planning algorithm with performance bounds. In *IEEE/RSJ International Conf. on Intelligent Robots and Systems*, 3260–3267.

Yu, J., and LaValle, S. 2013a. Planning optimal paths for multiple robots on graphs. In *IEEE International Conference on Robotics and Automation*, 3612–3617.

Yu, J., and LaValle, S. 2013b. Structure and intractability of optimal multi-robot path planning on graphs. In *AAAI Conference on Artificial Intelligence*, 1444–1449.

Table 2: Results on four-neighbor connected 20×15 grid maps for different numbers of robots and obstacle densities (given as averages over the instances solved within the ten-minute runtime limit). The number of exchange operations is given in parentheses. Black entries indicate that all instances were solved. Red entries indicate that only some instances were solved. Blue entries indicate that one or more solutions were suboptimal.

		Adaptive CBS		Optimal Flow			Suboptimal Flow		
Robots	Density	Solved	Time	Makespan		Time	Makespan		Time
10	0%	100.00%	0.03s	20.67	(0.70)	1.78s	20.67	(2.70)	0.55s
10	5%	100.00%	0.06s	20.77	(1.07)	1.61s	20.77	(2.37)	0.53s
10	10%	100.00%	0.12s	21.13	(0.83)	1.52s	21.13	(3.13)	0.54s
10	15%	100.00%	0.06s	22.37	(1.57)	1.65s	22.37	(3.30)	0.64s
10	20%	100.00%	0.19s	23.23	(2.23)	1.57s	23.23	(3.77)	0.63s
10	25%	100.00%	0.11s	24.70	(3.50)	1.52s	24.70	(4.97)	0.70s
10	30%	100.00%	0.53s	30.67	(5.37)	2.01s	30.73	(7.03)	1.02s
20	0%	100.00%	0.10s	23.67	(6.83)	5.20s	23.67	(9.87)	2.05s
20	5%	100.00%	0.28s	23.77	(5.77)	4.89s	23.77	(10.90)	2.09s
20	10%	100.00%	0.23s	24.07	(6.50)	4.61s	24.07	(11.67)	2.19s
20	15%	100.00%	0.23s	23.67	(7.80)	3.92s	23.67	(12.77)	1.87s
20	20%	96.67%	1.68s	25.20	(9.37)	3.88s	25.20	(15.20)	2.00s
20	25%	96.67%	1.51s	28.77	(13.27)	5.05s	28.77	(19.97)	2.95s
20	30%	73.33%	21.40s	33.03	(20.80)	7.20s	33.03	(26.30)	3.97s
30	0%	100.00%	1.24s	24.10	(13.70)	9.52s	24.10	(18.40)	5.41s
30	5%	100.00%	0.63s	25.63	(16.80)	10.81s	25.63	(22.10)	6.48s
30	10%	96.67%	0.44s	24.20	(15.30)	8.59s	24.20	(21.37)	4.73s
30	15%	100.00%	0.87s	24.97	(19.00)	8.07s	24.97	(24.47)	4.71s
30	20%	83.33%	3.11s	26.70	(23.60)	9.17s	26.77	(31.27)	6.91s
30	25%	53.33%	5.39s	30.27	(30.37)	22.16s	30.27	(38.73)	22.90s
30	30%	23.33%	112.27s	34.87	(41.53)	36.47s	34.87	(53.63)	35.46s
40	0%	96.67%	11.16s	25.13	(23.83)	23.78s	25.13	(31.93)	13.31s
40	5%	100.00%	15.47s	25.67	(27.77)	22.55s	25.67	(33.60)	15.09s
40	10%	96.67%	30.44s	25.40	(31.67)	20.76s	25.43	(38.53)	14.25s
40	15%	90.00%	90.10s	25.67	(34.83)	18.75s	25.67	(41.77)	16.35s
40	20%	53.33%	60.06s	25.33	(39.53)	22.27s	25.33	(44.70)	19.67s
40	25%	13.33%	118.03s	28.77	(46.20)	53.38s	28.77	(55.40)	61.20s
40	30%	0.00%	-	33.20	(68.83)	130.33s	33.20	(79.53)	92.78s
50	0%	96.67%	22.33s	26.20	(39.93)	78.38s	26.20	(46.50)	55.31s
50	5%	93.33%	7.22s	26.07	(40.93)	61.78s	26.07	(51.73)	49.57s
50	10%	80.00%	15.74s	25.40	(45.50)	46.91s	25.40	(54.00)	37.02s
50	15%	30.00%	61.46s	26.73	(55.43)	68.54s	26.73	(62.20)	56.52s
50	20%	16.67%	141.95s	27.50	(61.17)	89.05s	27.50	(67.53)	94.40s
50	25%	3.33%	378.24s	30.33	(75.50)	259.08s	30.37	(86.83)	268.41s
50	30%	0.00%	-	34.97	(105.53)	596.25 s	34.97	(123.83)	522.42s

Table 3: Results on benchmark game map brc202d for different numbers of robots (given as averages over the instances solved within the ten-minute runtime limit).

	Adaptive CBS		
Robots	Solved	Makespan	Time
5	100.00%	732.10	0.34s
10	100.00%	809.03	17.75s
15	96.67%	882.28	3.51s
20	86.67%	905.15	5.43s
25	96.67%	931.34	22.73s
30	86.67%	942.19	29.03s
35	76.67%	963.13	50.80s
40	53.33%	974.25	30.50s
45	70.00%	974.10	77.49s
50	36,67%	943.36	86.76s