Optimal Solutions to Large Logistics Planning Domain Problems – Detailed Proofs

Gerald Paul
Boston University
Boston, Massachusetts, USA
gerryp@bu.edu

Gabriele Röger and **Thomas Keller** and **Malte Helmert**
University of Basel
Basel, Switzerland
{gabriele.roeger,tho.keller,malte.helmert}@unibas.ch

Technical Report CS-2017-001
University of Basel, Department of Mathematics and Computer Science

Abstract

This report contains the proof of correctness for the multi-vehicle simplification presented in the paper *Optimal Solutions to Large Logistics Planning Domain Problems.*

1 Basic Definitions

For an introduction of the problem, we refer the reader to the main publication [Paul et al., 2017]. Here, we only briefly repeat the definitions that are necessary for specifying the theorem we want to proof.

1.1 Logistics tasks

Definition 1 (Logistics Task) *A* Logistics *task is given as a tuple* $\langle L, C, P, T, A, \mathit{city}, \mathit{airport}, \mathit{origin}, \mathit{dest}\rangle$*, where*

- L *is a finite set of* locations*,*
- C *is a finite set of* cities*,*
- P *is a finite set of* packages*,*
- T *is a finite set of* trucks*,*
- A *is a finite set of* airplanes*,*

- *city* : $L \to C$ *assigns each location a city,*
- *airport* : $C \to L$ *assigns each city an airport location in this city, i. e. city(airport(c)) = c for all* $c \in C$*,*
- *origin* : $P \cup T \cup A \to L$ *specifies the origin location of each package, truck and airplane, where the origin of an airplane is always an airport location, and*
- *dest* : $P \to L$ *defines a destination for each package.*

A *vehicle* is a truck or an airplane. A *state* s of a LOGISTICS task maps each vehicle v to a location $s(v)$ and each package p to a location, truck or airplane $s(p)$. The *initial state* is given by *origin*. There are four types of operators:

- Vehicles v can *load* packages p at the same location: $\mathit{load}(v, p, l)$ is applicable in state s if $s(v) = s(p) = l$ and leads to state s' that only differs from s in $s'(p) = v$.
- Vehicles v can *unload* loaded packages p: $\mathit{unload}(v, p, l)$ is applicable in state s if $s(p) = v$ and $s(v) = l$, and leads to state s' that only differs from s in $s'(p) = l$.
- Trucks t can *drive* to locations l in the same city: $\mathit{drive}(t, l)$ is applicable in state s if $\mathit{city}(s(t)) = \mathit{city}(l)$ and leads to state s' that only differs from s in $s'(t) = l$.
- Airplanes a can *fly* to all airport locations l: $\mathit{fly}(a, l)$ is applicable in state s if $l = \mathit{airport}(c)$ for some city c. The resulting state s' only differs from s in $s'(a) = l$.

A *plan* is a sequence of operators that are successively applicable to the initial state and lead to a state s_G with $s_G(p) = \mathit{dest}(p)$ for all packages $p \in P$. The *cost* of a plan is the length of the operator sequence. A plan is *optimal* if it has minimum cost among all plans.

We call packages that have the origin and the destination in the same city *intracity* packages and all other packages *intercity* packages. If a package p is in a vehicle at location l or it is directly at l, we refer to l as the *position* $\mathit{pos}_s(p)$ of p in state s; formally, $\mathit{pos}_s(p) = s(p)$ if $s(p) \in L$ and $\mathit{pos}_s(p) = s(s(p))$ if $s(p) \in T \cup A$. We use the term *region* to denote all locations of a city (a *truck region*) or all airports (a *plane region*). The vehicles for a given region are the corresponding trucks for a truck region, or the airplanes for the plane region.

1.2 Delivery Graphs

An edge $l \to l'$ in a *delivery graph* for a region represents the information that a package needs to be transported from location l to location l' in this region. We distinguish airplane and truck delivery graphs.

Definition 2 (Airplane Delivery Graph) *For state s of logistics task $\langle L, C, P, T, A, \mathit{city}, \mathit{airport}, \mathit{origin}, \mathit{dest}\rangle$, the* airplane delivery graph *is the directed graph $D_s^A = (C, E)$, where $E = \{(c, c') \mid$ there is a $p \in P$ s.t. $c = \mathit{city}(\mathit{pos}_s(p)) \neq \mathit{city}(\mathit{dest}(p)) = c'\}$.*

Definition 3 (Truck Delivery Graph) *For state s of logistics task $\langle L, C, P, T, A, \mathit{city}, \mathit{airport}, \mathit{origin}, \mathit{dest}\rangle$ and city $c \in C$, the* truck delivery graph for c *is the directed graph $D_s^c = (V, E)$, where*

- *$V = \{l \in L \mid \mathit{city}(l) = c\}$ are the locations in city c, and*
- *E contains the following edges for each package p with $\mathit{pos}_s(p) \neq \mathit{dest}(p)$:*
  - *If $\mathit{city}(\mathit{pos}_s(p)) = \mathit{city}(\mathit{dest}(p)) = c$ then there is an edge $\mathit{pos}_s(p) \to \mathit{dest}(p)$.*
  - *If $\mathit{city}(\mathit{pos}_s(p)) = c$, $\mathit{city}(\mathit{dest}(p)) \neq c$ and $\mathit{pos}_s(p) \neq \mathit{airport}(c)$ there is an edge $\mathit{pos}_s(p) \to \mathit{airport}(c)$.*
  - *If $\mathit{city}(\mathit{pos}_s(p)) \neq c$, $\mathit{city}(\mathit{dest}(p)) = c$ and $\mathit{dest}(p) \neq \mathit{airport}(c)$ there is an edge $\mathit{airport}(c) \to \mathit{dest}(p)$.*

2 Multi-vehicle simplification: Proof idea

We want to proof the following theorem from the paper [Paul et al., 2017]:

Theorem 2 *Consider a solvable* LOGISTICS *task where each delivery graph has exactly one non-trivial weakly connected component, i.e., one weakly connected component plus zero or more isolated locations.*[1] *Then there is an optimal plan using one truck from each city and one airplane.*

Proof strategy: take a given plan (for example an optimal one) that uses multiple vehicles in one region and rewrite it to use fewer vehicles without increasing its cost. If this is always possible, we never need multiple vehicles per region.

We try to study this in a "factored" way, by focusing on one region at a time. We look for the following, more restricted class of transformations: given a plan and a region where multiple vehicles are used, modify the plan so that

1. its cost does not increase,
2. it uses fewer vehicles for the selected region, and
3. the subsequence of actions using vehicles that do not belong to the selected region remains the same.

[1]Isolated locations are ones where no package must be unloaded or loaded. They may serve as starting locations of vehicles, but are otherwise of no use.

In other words, we are only allowed to remove/insert/reorder actions involving the vehicles of the selected region. Let's consider the search for such a modification the "replanning problem" for the selected region.

We can think of a replanning problem as a classical planning problem which is a logistics problem with a single region with additional actions that encode the "outside influence" in the given plan. There are two kinds of such additional actions: $\mathit{arrive}(p, l)$ means that package p arrives at the region at location l (i. e., it is dropped there by a vehicle that does not belong to the region in the given plan), and $\mathit{leave}(p, l)$ means that package p leaves the region at location l. Constraint 3 above implies that in the replanning problem, the sequence of arrive/leave actions is fixed, so we have to find a plan that is compatible with the given arrive/leave sequence.

Rather than insisting on exactly the given order of arrive/leave actions we can be a bit more flexible: by making a package arrive later than it actually arrives in the given plan or by making a package leave earlier than it actually leaves in the given plan, we never lose compatibility with the given plan. This means that alternatively we can represent the outside plan with the information which packages arrive and leave at which locations, plus constraints of the form $\mathit{leave}(p_1, l_1)$ must occur before $\mathit{arrive}(p_2, l_2)$ that specify that we must finish delivering a certain package before we can begin delivering a certain other package. Other constraints are not necessary. We could also phrase this as "p_1 must be delivered before p_2" with the meaning that the delivery of p_1 must *finish* before the delivery of p_2 may *begin*. The replanning problem can thus be viewed as a single-vehicle logistics problem with delivery order constraints of this form.

3 Truck problems

Consider a replanning problem involving a truck region. We call this a "truck problem". These are simpler than replanning problems involving a plane region ("plane problem") because all arrivals/leaves happen at the same location (the airport).

Consider a plan for a truck problem (with weakly connected delivery graph) using two trucks. We associate each action with a truck, including *arrive* and *leave* actions, which we associate with the truck that delivers the package within the city. (We can ignore the case where the package never needs to be moved after arriving/before leaving because then there is nothing to do, and of course we also know that in an optimal plan we can assume that only one vehicle picks up and drops a package that needs to be moved, and it is picked up and dropped only once.)

Each action then belongs to exactly one truck. We call load/unload/move actions *private* because no other truck depends on them nor do they depend on other trucks and arrive/leave actions *public* because such dependencies exist (because we must satisfy the given arrive/leave constraints).

Two private actions of different trucks always commute: if two private actions for different trucks are adjacent in a plan, then we can swap their order. A public action of one truck and a private action of another truck also commute.

Consider a plan including two trucks T_1 and T_2 that visit at least one common location L. (We say a truck *visits* a location if it is ever locate there, including in the initial state.) Such trucks and such a location must exist if the delivery graph is weakly connected and two trucks are used. (It is OK if the plan uses further trucks besides T_1 and T_2.)

First, consider the case where at least one of them (w.l.o.g. T_1) never visits the airport. Then T_1 never applies a public action. We can thus move all its actions to the end of the plan, as they do not interact with other trucks. If the common location L is not the initial location of T_2, it is easy to see that the role of T_1 can be taken over by T_2: let the sequence of visited locations of T_1 be Z, and let the sequence of visited locations of T_2 be $X_2 + [L] + Y_2$. Then replace the visit to $[L]$ by the movement sequence Z, i.e., make T_2 visit $X_2 + Z + Y_2$, doing T_1's job on the way.

If L is the initial location of T_2 but not of T_1, then make T_1 take over the job of T_2 instead: move all actions of T_1 up to visiting L (and loads/unloads done there) to the front of the plan and move all actions of T_1 after visiting L to the end of the plan. Then replace T_2 with T_1 everywhere.

If L is the initial location of both, directly replace T_2 with T_1 everywhere.

This leaves the case where there are two trucks visiting the airport. This means that there exist two trucks performing public actions. Let T_1 and T_2 be the **first** trucks that apply a public action. Reorder all private actions so that at the beginning of the plan are all movements of T_1 until just before its first public action, followed by all movements of $_T2$ just before its first public action, followed by public actions of T_1 and T_2 at the airport, followed by the rest of the plan. Then either T_1 or T_2 (depending on who, if anyone, visits each other's location at this prefix of the plan) can take over the others job at the prefix of the plan as in the previous case, and after visiting the airport, this truck can continue the remaining job of the other truck. We can then get rid of the other truck.

4 Plane problems

Because package interchanges with the "outside" can happen in every location in plane problems (rather than just at the single airport of a truck problem), plane problems are involved. The remaining sections of this proof show how to address the replanning problem for planes so that every weakly connected component of the plane region only uses one plane.

4.1 Problem considered

We consider the *airplane problem*, which is defined by

- a finite set of locations L
- a finite set of vehicles (airplanes) V

- a finite set of packages P
- an initial location $\mathit{init}(p)$ for each package
- a goal location $\mathit{goal}(p)$ for each package
- an initial location $\mathit{init}(v)$ for each vehicle
- a precedence relation $\prec \subseteq P \times P$,
  where $p \prec q$ implies $\mathit{goal}(p) = \mathit{init}(q)$ and $\prec$ is acyclic.

The semantics is defined as usually for Logistics tasks, with the additional restriction that whenever $p \prec q$, q may only be picked up at its initial location after p has been dropped at its goal location.

4.2 Restrictions on instances

We can make a number of further restrictions on these instances without loss of generality:

- L, V and P are all non-empty.
- $\mathit{init}(p) \neq \mathit{goal}(p)$ for all packages
- The delivery graph (the digraph over L with arc set $\{\langle \mathit{init}(p), \mathit{goal}(p)\rangle \mid p \in P\}$) has exactly one non-trivial weakly connected component (i.e., one weakly connected component with 2 or more vertices, plus possible isolated vertices).
- The delivery graph has at most one isolated vertex, and if it does, then it must be the initial location of some vehicle.

Note: the delivery graph contains an arc for p whether or not there is a vehicle initially located at $\mathit{init}(p)$ (or $\mathit{goal}(p)$). So the delivery graph should not be confused with the landmark ordering digraph used to compute the feedback vertex set heuristic.

4.3 Restrictions on plans

When considering optimal plans, we can assume that every package is loaded exactly once and unloaded exactly once.

Because each package must be moved, this means it is transported by exactly one vehicle. For a given plan, we write $\mathit{vehicle}(p)$ for the vehicle that transports package p.

We write the actions of the plan as:

- $\mathit{move}(v, l, l')$: move vehicle v from location l to location l'; we may omit v and/or l where clear from context

- $\mathit{pickup}(v,p,l)$: use vehicle v to load package p at location l; we may omit v and/or l where clear from context (must have $v = \mathit{vehicle}(p)$ and $l = \mathit{init}(p)$)
- $\mathit{drop}(v,p,l)$: use vehicle v to unload package p at location l; we may omit v and/or l where clear from context (must have $v = \mathit{vehicle}(p)$ and $l = \mathit{goal}(p)$)

Many pairs of actions a and a' are commutative, i.e., when they appear next to each other in a plan, their order can be swapped without affecting the correctness of the plan. Sometimes, they are only semi-commutative in the sense that plans of the form $\pi a a' \sigma$ (where π is an arbitrary prefix and σ is an arbitrary suffix) can be transformed to $\pi a' a \sigma$, but the opposite transformation is not always correct.

(In general, a sufficient condition for this transformation to be allowed is that a never enables a', a' never disables a, and the combined effect of aa', ignoring preconditions, is the same as the combined effect of $a'a$).

In particular, the following semi-commutativity transformations are valid:

- $\pi \mathit{pickup}(v,p,l) a' \sigma$ can be transformed to $\pi a' \mathit{pickup}(v,p,l) \sigma$ for all actions a that are not of the form $\mathit{move}(v,l,l')$.

  In other words, a pickup action of vehicle v can always be demoted by one step unless it is followed by a movement of v.

  Note that the opposite transformation is not always correct because it may introduce the violation of a precedence constraint.

- $\pi a \mathit{drop}(v,p,l) \sigma$ can be transformed to $\pi \mathit{drop}(v,p,l) a \sigma$ for all actions a that are not of the form $\mathit{move}(v,l',l)$.

  In other words, a drop action of vehicle v can always be promoted by one step unless it is preceded by a movement of v.

  Note that the opposite transformation is not always correct because it may introduce the violation of a precedence constraint.

It is also easy to see that one or more steps after a pick-up action of vehicle v there must be a movement of vehicle v (because the picked-up package must be dropped somewhere else), and one or more steps before a drop action of vehicle v there must be a movement of vehicle v (because the dropped package must have been picked up somewhere else).

4.4 Extended moves

The discussion in the previous section implies that we can permute the actions in every plan so that it is a sequence of "extended moves", where each extended move involves a vehicle v and locations l and l' and is of the following form:

- zero or more actions $\mathit{pickup}(v,p_1,l), \ldots, \mathit{pickup}(v,p_m,l)$

- the action $move(v, l, l')$
- zero or more actions $drop(v, q_1, l'), \ldots, drop(v, q_n, l')$

To do this, just keep demoting every pickup action of vehicle v not followed by another pickup action of v or movement of v until no more such transformations are possible. Similarly, keep promoting every drop action of a vehicle v not preceded by another drop action of v or a movement of v until no more such transformations are possible. At the end of this process, the plan will have the required form. Call plans of this form in *normal form*.

In plans in normal form, packages are always picked up "as late as possible", i.e., just before leaving a location, and dropped "as early as possible", i.e., just after entering a location. (Here, "as late/early as possible" should not be taken too literally. It is only meant relative to the next/preceding movement of the vehicle. In some plans, certain packages could be handled earlier or later by handling them during an earlier/later visit of the same location.)

We can then sufficiently describe a plan by a sequence of extended move actions, and we do not need to mention which packages are picked up and dropped by the extended move: always picking up/dropping *all* permitted packages works. (Note that this is not as easy as picking up all undelivered packages assigned to v with $init(p) = l$ because picking up a package may be forbidden by a precedence constraint. Determining which packages and actually picked up and dropped at which point in the plan requires more global reasoning, which is why we introduce additional notation for this in the following section.)

We write an extended move involving the movement $move(v, l, l')$ as $Move(v, l, l')$ and may omit v and l where clear from context. If the k-th step of a sequence of extended moves representing a plan is $Move(v, l, l')$, we say that v leaves l at time step k and enters l' at time step k. Of course vehicles may leave or enter locations multiple times.

4.5 Delivery times and collection times

We assume a plan is given as a sequence of extended moves. To see whether such a plan works, it is useful to also specify information about the *collection times* $collect(p)$ and *delivery times* $deliver(p)$ of the packages.

For a given plan, we define $collect(p) = k$ if p is picked up as part of the k-th extended move in the sequence, and $deliver(p) = k$ if p is dropped as part of the k-th extended move in the sequence.

Assume we are given a plan in the following representation:

- the sequence $moves = \langle M_1, \ldots, M_n \rangle$ of extended moves, each move represented in the form $Move(v, l, l')$.
- the assignment of packages to vehicles, $vehicle(p)$ for each package
- the collection time $collect(p)$ and delivery time $deliver(p)$ for each package

Such a plan is correct iff the following conditions hold:

C1. The movement sequence for every vehicle is consistent, i.e., if v moves at all in the plan, then its first move must begin at $\mathit{init}(v)$ and all subsequent moves of v must begin at the location the previous move of v ended.

C2. Packages are collected before they are delivered: $\mathit{collect}(p) < \mathit{deliver}(p)$ for all packages p.

C3. The precedence constraints are met: $\mathit{deliver}(p) < \mathit{collect}(q)$ whenever $p \prec q$.

C4. The collection times are consistent with the vehicle movements: given package p, let $v = \mathit{vehicle}(p)$, $t = \mathit{collect}(p)$ and $l = \mathit{init}(p)$. Then $M_t = \mathit{Move}(v, l, l')$ for some location l', i.e., in the t-th step, vehicle v leaves the initial location of p.

C5. The delivery times are consistent with the vehicle movements: given package p, let $v = \mathit{vehicle}(p)$, $t = \mathit{deliver}(p)$ and $l = \mathit{goal}(p)$. Then $M_t = \mathit{Move}(v, l', l)$ for some location l', i.e., in the t-th step, vehicle v enters the goal location of p.

It is important to realize that this is an "if and only if" relationship: we can represent every plan in normal form in terms of *moves*, *vehicle*, *collect* and *deliver* such that C1-C5 are satisfied, but also conversely, if we are given *moves*, *vehicle*, *collect* and *deliver* satisfying C1-C5, then they induce a valid plan. The cost of the plan is of course the length of the movement sequence plus two times the number of packages, so we need to minimize the length of *moves*. Hence all transformations that do not increase the length of *moves* do not affect optimality.

Idea: given an optimal plan that might use multiple vehicles, first compute *moves*, *vehicle*, *collect* and *deliver* for the given plan, then compute a new plan represented by moves', $\mathit{vehicle}'$, $\mathit{collect}'$ and $\mathit{deliver}'$ using only one vehicle, and then convert this to a sequence of move/pickup/drop actions.

4.6 Limited case

We begin with the limited case where all airplanes are initially located at an isolated location. This implies that there is nothing useful that can be done at the initial location of the airplanes and that in an optimal plan no airplane will ever fly to this location.

Assume we are given an optimal plan in the form of *moves*, *vehicle*, *collect* and *deliver*. A first attempt at transforming this plan into a plan represented by moves', $\mathit{vehicle}'$, $\mathit{collect}'$ and $\mathit{deliver}'$ using only one vehicle might be as follows:

- Select a single vehicle v.
- Set $\mathit{vehicle}'(p) = v$ for all packages p.

- Set moves' to be the movement sequence where in the k-th step, v moves to the same location that is the target location in the k-th step of moves. In other words, moves' moves to the same sequence of locations as moves, in the same order, but all movements are made by v. (This sequence may include movements from a location to itself, but that is not a problem although if this happens, the original plan was not optimal.)
- Set $\mathit{collect}'(p) = \mathit{collect}(p)$ for all packages p.
- Set $\mathit{deliver}'(p) = \mathit{deliver}(p)$ for all packages p.

If we did not have precedence constraints, this strategy of just keeping the plan "as-is" and letting a single vehicle carry out all movements in the same sequence as in the given plan would work. However, with precedence constraints it does not work in general.

The reason why it doesn't work is that this assignment may violate condition C4 above. The other conditions C1, C2, C3, C5 are satisfied. The reason why C4 is violated is that vehicle v does not necessarily *leave* the same location in the k-th step of moves' that the vehicle that moves in the k-th step of moves leaves.

To repair this, we set $\mathit{collect}'(p)$ to a time point which is (potentially) different from $\mathit{collect}(p)$. We must do this in such a way that C2, C3 and C4 are satisfied for the new solution. (C1 and C5 do not depend on $\mathit{collect}'$.)

Let $l = \mathit{init}(p)$ and let $t = \mathit{collect}(p)$. Set $\mathit{collect}'(p)$ to the *largest* (latest) time point $t' \leq t$ at which vehicle v leaves l in moves'. Such a time point must exist: in moves, some vehicle v' leaves l for some location $l' \neq l$ at t, which implies that v' must have entered l at some previous time point $t'' < t$. This means that in moves', v enters l at time step t'' (possibly from l itself, which is no problem), so is located at l after step t''. Moreover, v enters $l' \neq l$ in moves' at time t, so is located at a different location from l after step t. This implies that v must leave l in moves' at some time $t' \leq t$.

With this definition of $\mathit{collect}'$, C4 is satisfied. C2 remains satisfied because $\mathit{collect}'(p) \leq \mathit{collect}(p)$ for all packages p (due to the restriction $t' \leq t$ in the previous paragraph).

The question is whether C3 is still satisfied, as reducing the values of $\mathit{collect}$ makes these constraints tighter. So we must verify $\mathit{deliver}'(p) < \mathit{collect}'(q)$ for all precedence constraints $p \prec q$.

Let $p \prec q$ be such a constraint, and let $l = \mathit{goal}(p) = \mathit{init}(q)$ be the location involved. In the original plan, say we have $\mathit{vehicle}(p) = v_p$ and $\mathit{vehicle}(q) = v_q$.

Let's say $\mathit{deliver}(p) = t_1$ and $\mathit{collect}(q) = t_2$. We have $t_1 < t_2$ because the original plan is correct. We have $\mathit{deliver}'(p) = t_1$.

So in the original plan, v_p moves from another location to l at time t_1, which implies that in the new plan v moves to l at time t_1 (possibly not from another location but from l itself, but this does not matter).

Furthermore, in the original plan v_q moves from l to some other location l' at time t_2, which implies that in the new plan v moves to l' at time t_2 (possibly from l' itself, but again this does not matter). This means that in the new plan v is at l at time t_1 and at l' at time t_2, so must leave l at one or more time steps in the range $\{t_1 + 1, \ldots, t_2\}$. We have defined $\mathit{collect}'(q)$ as the latest time point $t' \leq \mathit{collect}(q) = t_2$ at which v leaves $\mathit{init}(q) = l$, so we must have $t_1 + 1 \leq \mathit{collect}'(q) \leq t_2$.

Putting this together with $\mathit{deliver}'(p) = t_1$, this proves $\mathit{deliver}'(p) = t_1 < t_1 + 1 \leq \mathit{collect}'(q)$, i.e., $\mathit{deliver}'(p) < \mathit{collect}'(q)$, concluding the proof.

4.7 Reducing the general case to the restricted case

We now consider the general case where airplanes can start at arbitrary locations.

Consider an optimal solution π using $k \geq 2$ airplanes for the given problem instance Π. We want to construct a plan for Π that only uses one of the k airplanes and has the same cost as π.

Consider a modified problem Π' where the k airplanes begin at a new isolated location. The modified problem then satisfies the requirements of the restricted case. We obtain a plan π' for the modified problem by inserting, at the start of π, one action for each of the k airplanes to fly it to its initial location in Π. Clearly, π' consists of k more actions than π.

We can apply the transformation for the restricted case to Π' and π' to obtain a new solution π'' for Π' that only uses one airplane v. Moreover, all moves in π'' will only fly to locations that also exist in Π, so if we change the first action of π'' to originate from the original $\mathit{init}(v)$ rather than the new isolated location, π'' is also a plan for the original problem Π.

When generated in this way, π'' will be k actions too long. However, π'' may contain actions of the form $\mathit{move}(v, l, l)$, which are redundant and can be omitted. The key to the generalization will be permuting π' before we perform the transformation from π' to π'', in such a way that π'' will contain at least k redundant actions.

4.7.1 Permuting the plan to obtain redundant actions

Recall that the movement path of v in π'' is simply the sequence of locations visited by any of the airplanes in π', in the same order as the actions are executed in π'. So we want to permute π' in such a way that there are two consecutive movements (by different vehicles) to the same location as often as possible.

We will create $k - 1$ such situations, saving us $k - 1$ actions. The remaining action will be saved by observing that the first move in π' will move (from the artificial initial location) to the initial location of some vehicle. By making this vehicle our vehicle of choice in the one-vehicle plan for the original problem, we can save another action. The observation that the first action is of this form holds because in all our permutations below, we never permute the relative order of locations visited by a given vehicle, only the order in which different vehicles act, and hence the first overall movement must be the first movement of one of the vehicles (which moves it from the artificial initial location to its original initial location).

We construct a permuted plan from π' (viewed as a plan consisting of extended actions of the form Move(v, l, l')) as follows:

- We process the actions in the original plan step by step.
- Initially, partition the vehicles into k groups, where each group consists of exactly one vehicle. As we process actions, groups can be merged with others to reflect the fact as the subplans for the vehicles in the groups begin to interact.
- The invariant of the algorithm is that while two vehicles are in different groups G and G', the sets of locations that have been moved to so far by vehicles in G and G' are disjoint. In particular, this means that at every stage of the algorithm each location is either "owned" by exactly one group (meaning a vehicle from this group, and no other group, has visited it) or "unowned" (meaning that no vehicle has visited it yet).
- This disjointness of vehicles and locations implies that the subplans considered so far for different groups don't interact and can be interleaved arbitrarily. We exploit this by keeping track of the new plan generated by the algorithm as a collection of subplans, one for each group, without committing early how to interleave these subplans.
- Every time the next step in the input plan involves a vehicle moving to a location owned by a different group, the groups must be merged. At this point we combine their subplans and arrange for a suitable interleaving that allows us to save an action.
- At the end of the execution we must have a single group. Otherwise we would still have multiple location-disjoint subplans for different vehicle groups at the end, which would imply that the delivery graph is not weakly connected.
- Because we go from k groups to 1 group and save an action every time we combine two groups into one, we save $k-1$ actions altogether.

Input:

- pi' = the input plan
- V = the vehicles used in the input plan pi'
- L = the set of locations

Algorithm:

```
def main():
    groups = {{v} | v in V}
    for each group in groups:
group_plans[group] = empty plan
```

```
    for each location l in L:
owner[l] = undefined
    for each extended action Move(v, l, l') in pi', in order:
# Assertion: if l is not the artificial start location,
# then the owner of l is the group of v.
group_of_v = groups[v]
if owner[l'] = undefined:
    owner[l'] = group_of_v
    group_plans[group_of_v].append(Move(v, l, l'))
else if owner[l'] = group_of_v:
    group_plans[group_of_v].append(Move(v, l, l'))
else:
    # l' is already owned by someone else, i.e., some vehicle
    # from a different group has previously visited l'. We
    # must marry the group of v to the group owning l'.
    current_owner_group = owner[l']
    # Marry current_owner_group and group_of_v:
    # 1. Create combined group and update set of groups.
    combined_group = current_owner_group \cup group_of_v
    groups.remove(current_owner_group)
    groups.remove(group_of_v)
    groups.add(combined_group)
    # 2. Combine subplans for the two groups.
    subplan1 = group_plans.pop(current_owner_group)
    subplan2 = group_plans.pop(group_of_v)
    group_plans[combined_group] = combine_plans(
subplan1, subplan2, v, l, l')
    # 3. Update owner information to reflect new group
    for all l'':
if owner[l''] = current_owner_group:
   owner[l''] = combined_group
if owner[l''] = group_of_v:
   owner[l''] = combined_group

def combine_plans(plan_of_prev_owner_of_l', plan_of_v, v, l, l'):
    # Invariants:
    # - plan_of_prev_owner_of_l' and plan_of_v refer to disjoint
    #   sets of vehicles and locations.
    # - plan_of_prev_owner_of_l' contains at least one action
    #   of the form move(*, *, l').
    # - plan_of_prev_owner_of_l' contains no actions involving v
    #   or l; plan_of_v may contain actions involving v and l, but
    #   does not have to (if this is the first movement of v).
    #
    # Consequences:
    # 1) All actions of prev_owner_of_l' and plan_of_v
#    commute, so we can interleave them arbitrarily.
# 2) Move(v, l, l') commutes with all actions of prev_owner_of_l'
#    that do not involve l'.
Split plan_of_prev_owner_of_l' into prefix + suffix such that
   the last action in prefix is of the form move(*, *, l')
   and no action in suffix is of this form.
return plan_of_v + prefix + [move(v, l, l')] + suffix
# Note that this contains two adjacent movements to l', as
# the last action of prefix moves to l' and so does move(v, l,
# l').
#
# Note also that if one of the input plans already contained such
# adjacent movements, the rescheduling that happens in this
# function does not break them apart (which might lose an
# opportunity to save an action): plan_of_v remains contiguous, so
# any subsequent moves to the same location are preserved, and
# plan_of_prev_owner_of_l' is broken apart into prefix + suffix in
# such a way that the last action of prefix moves to l' and no
# action of suffix moves to l', so subsequent moves to the same
# location within plan_of_prev_owner_of_l' must either be fully
# within prefix or fully within suffix.
```

References

[Paul et al., 2017] Paul, G., Röger, G., Keller, T., and Helmert, M. (2017). Optimal solutions to large logistics planning domain problems. In Fukunaga, A. and Kishimoto, A., editors, *Proceedings of the 10th Annual Symposium on Combinatorial Search (SoCS 2017)*. AAAI Press.