Ambulance planning with distance optimization

Team 14

Federico Baldassarre {fedbal@kth.se}, Vladislav Polianskii {vpol@kth.se},
Ardhendu Shekhar Tripathi {tripa@kth.se}, Mónica Villanueva Aylagas {monicavi@kth.se}

Abstract

Optimizing task distribution between multiple agents can prove hard for a human operator, especially when route planning and resource constraints are also involved.

Our project focuses on the automation of an emergency medical center that receives calls from patients with varying severity and dispatches ambulances to drive them to an hospital. In such a situation the constraints and the metrics considered are the number of vehicles, the priority assigned to each patient and the travel distances. In addition, we also focus on positioning free ambulances so that the travel distance to the nearest nodes is minimized, accounting also for the demand that every node expresses, i.e. the probability of an emergency to arise there.

We propose a *K-means* inspired algorithm for the maximum coverage allocation of the ambulances, which also uses the *Dijkstra* algorithm to find the shortest paths. The *Hungarian Algorithm* and a modified version of the *Particle Swarm Optimization* algorithm are used to generate plans for the ambulances optimizing the highest patient severity and the minimum traveling cost. The performances and the plans generated by these two approaches are then compared on situations that differ by the number of city nodes, roads, patients and ambulances. The cases where each planner is expected to work better are also explored.

1 Introduction

The problem is based on the routing of ambulances over different emergencies around a city. Emergencies can appear at any moment, at every location following a probability distribution. The goal of the planner is to allocate the best ambulance to serve an emergency and the route it has to follow to pick up the patient and drop it at a hospital. This is done by taking into account the severity of the case and the distance between locations. To fully understand the problem it is required to define some terms and clarify certain assumptions.

The city is modeled as a non-directed graph so that each node represents a location and the roads are bidirectional, meaning that we do not account for one way roads. Every location bear a demand, a statistical measure of the propensity of an emergency to appear there, the demands are constant over time, meaning that we ignore that certain locations can be more or less exposed to risk during certain hours. Likewise, the severity of a patient is measured on a scale from 1 to 3 and it does not change with time, i.e. the patient remains stable until carried to the hospital and can not die.

Along with this, it is assumed that an ambulance can carry only one patient at a time and an every hospital can always deal with every patient, meaning that we do not take into account different types of emergency or the capability of the hospitals. Cars give way to ambulance so it can be assumed that the travel cost is fixed between two locations. When measuring the cost of travel we only consider the distance and not the time, in other words actions of movement between to nodes are atomic. Also, the graph is fully connected, so it is possible to reach every location of the city from any other one. Finally, we do not account for unforeseen problems like a vehicle breakdown.

In order to account for the constraints described, it is necessary to use a more powerful version of PDDL. Version 2.1 has introduced the concepts of metric optimization, numerical functions and durative actions [1]. Unfortunately, the literature focuses mainly on the theoretical description and formal definition of the language, failing to propose planners able to solve PDDL2.1 problems. For this reason we decided to design and implement a domain specific planner able to solve a simplified version of the ambulance world problem. Nonetheless, the solution proved to be non trivial and involved understanding and using a set of graph analysis algorithms.

1.1 Problem formulation

To express the problem in PDDL it is necessary to define a domain and a problem. The former statically describes the types of objects, predicates, functions and actions used in the context of all the problems associated to the domain. The latter includes represent a specific instance of a problem to be solved, including the instantiation of objects, the initial state, the desired goal and possibly a metric to minimize/maximize.

Here follows a brief description of the domain and the problem, while the complete files can be found in the appendix [5.1].

Domain: Ambulance world

To begin with, it is necessary to specify the types of objects that will appear in this world: Ambulance, Hospital, Patient and Location.

The predicates are properties of the objects (boolean-valued functions). The following predicates are defined for our world:

ROAD(Location a, Location b): A road exist between two locations, that is, the both locations are reachable for each other.

AT(Patient|Ambulance|Hospital, Location): A patient, ambulance or hospital is situated in a certain location.

IN(Patient, Ambulance): A patient is inside an ambulance.
AVAILABLE(Ambulance): An ambulance is available to pick a patient.
WAITING(Patient): A patient is currently waiting for an ambulance.
INHOSPITAL(Patient): A patient is currently in a hospital.

One of the powerful features newly introduced by PDDL 2.1 are predicates, i.e. functions that return a numerical value for a given object in the domain. These values can be used to express the metrics and constrains to be taken into account by the planner.

LOCATIONCOORD(Location): Returns the Cartesian coordinates in which the location is situated.
LOCATIONDEMAND(Location): Returns the demand of a location. This is a statistical parameter calculated in cities so that the higher the number is, the higher the probability of having patients there.
DISTANCE(Location a, Location b): Returns the distance between two locations.
PRIORITY(Patient): Returns the priority of the patient, where priority $\in \{1, 2, 3\}$

It is also necessary to define valid actions for the domain, in terms of objects involved in the action, preconditions and effects:

MOVE(Ambulance a, Location from, Location to): An ambulance moves from a location to another.
PRECOND: ROAD(from, to) $\wedge$ AT(a, from)[1]
EFFECT: $\neg$AT(a, from) $\wedge$ AT(a, from)
PICK(Patient, Ambulance, Location): A patient is picked up by an ambulance.
PRECOND: WAITING(Patient) $\wedge$ AVAILABLE(Ambulance) $\wedge$ AT(Patient, Location) $\wedge$ AT(Ambulance, Location)
EFFECT: $\neg$WAITING(Patient) $\wedge$ $\neg$AVAILABLE(Ambulance) $\wedge$ IN(Patient, Ambulance)
DROP(Patient, Ambulance, Hospital, Location): A patient is dropped by an ambulance in a hospital.
PRECOND: IN(Patient, Ambulance) $\wedge$ AT(Ambulance, Location) $\wedge$ AT(Hospital, Location)
EFFECT: INHOSPITAL(Patient) $\wedge$ $\neg$IN(Patient, Ambulance) $\wedge$ AVAILABLE(Ambulance)

Problem

The description of a general problem can be separated in 3 parts: the definition of the objects, their properties and their locations.

$$\forall i PATIENT(p_i | i \in [1, \ldots, P]) \wedge \forall j AMBULANCE(a_j | j \in [1, \ldots, A]) \wedge \forall k HOSPITAL(H_k | k \in [1, \ldots, H]) \wedge \forall l LOCATION(L_l | l \in [1, \ldots, L]) \wedge \forall m \exists n ROAD(l_m, l_n | m, n \in 1, \ldots, L)$$

In this case, the properties of the objects are defined using functions. For Location objects we define the Cartesian Coordinate function and the Demand function. For Patient objects we define a Priority function. For pairs of connected locations we define a Road function yielding the cost of travel.

$$\forall l \exists x \exists y LOCATIONCOORD(l_l) = (x, y) \wedge \forall l \exists dem LOCATIONDEMAND(l_l) = dem \wedge \forall m \exists n \exists dist DISTANCE(l_m, l_n) = dist \wedge \forall i \exists pr PRIORITY(p_i) = pr$$

To locate every object on the map we define an At predicate.

$$\forall i \exists l AT(p_i, l_l) \wedge \forall j \exists l AT(a_j, l_l) \wedge \forall k \exists l AT(h_k, l_l)$$

The goal and the metric in out problem is fixed. The goal is taking all the patients to a hospital and the metric is minimizing the distance traveled by the ambulances as well as serving the number

[1]To reduce the notation, it is here assumed that the objects match the type declared in the parameters

of patients with highest priority first.

2 Related literature

The specific problem of Emergency Medical Service (EMS) planning has already been studied in literature. Providing care with minimum delay is crucial in this case.

Earlier attempts in this domain were done by Talarico *et al.* [2] who struggled to solve the ambulance routing problem at a catastrophe with high number of personal damage using a large neighborhood search metaheuristic. Knyaskov *et al.* [3] study the dynamic nature of a city with changing factors using real statistics to solve the problem in St. Petersburg, Russia. There have also been extensive studies [4] [5] on the shortest path calculations for routing which explore different strategies of finding the optimal path for an ambulance to reach the patient.

The EMS planning problem, being a classic example of multi agent coordination, has been subjected to different approaches. Multi agent centralized planning is discussed in detail under temporal constraints and uncertainty of admissible operations by Baki *et al.* [6]. Inspiration was taken from the previous work to ease the complexity of dealing with several units.

In order to provide optimal service to the patients, it is important that the ambulances are positioned in such a way that they maximize the expected coverage of demands. The *Maximum Covering Location Model* has been used extensively in analyzing locations for public service facilities. Attempts have been made to solve this problem for deciding optimal facility locations. Examples include maximal set covering algorithm [7], p-median and p-center problems [8] [9]. The present work focuses on a clustering inspired approach for solving this problem of ambulance location allocation for maximal coverage.

The path optimization problem stated in the present work can be viewed as a variant of a vehicle routing problem. It has many different meta-heuristic solutions, like ant colony [10] or genetic algorithms [11]. One of the solutions that comes as an inspiration for one of the methods (section 3.1) used in this work, was made by Wenbin Hu *et al.* [12]. In the article, authors use a modified version of particle swarm optimization algorithm for a vehicle routing problem with time windows. They use a chaos algorithm combined with Gaussian mutation to reinitialize the swarm when it falls into a local minimum.

3 Solution

In the following section we describe the different algorithms implemented for planning the ambulance dispatch problem as well as idle vehicles reorganization.

The study cases and its empirical results are also accounted at the end of this section.

3.1 Approaches and methods

Automatic problem generation: Erdös–Rényi

In order to test the implemented algorithms and analyze how they behave under different conditions we have developed a PDDL problem generator. Given some parameters such as the size of the map (in terms of nodes and roads), the number of patients, hospitals and ambulances, etc.

To ensure the connectivity of the graph the *Erdös–Rényi* algorithm is used, adding a slight modification that allows to insert as many edges as requested. The generator outputs a PDDL problem file such as the one described above.

The basic Erdös–Rényi algorithm start with two connected nodes. To ensure full connectivity, the next node is attached randomly to one of the previous ones. At the end of the iterations, the graph has N nodes and N-1 edges.

In order to use the desired number of edges and create multiple paths between nodes, edges are randomly added by two unconnected nodes.

Planner

The main component of the software is then able to parse the cities described in the PDDL files in a more versatile internal representation based on adjacency matrix. It then runs one one of the two planning algorithms devised and executes the plan yielded in a step-based manner. For every step it selects the actions that can be run concurrently, i.e. the first action of every ambulance, and executes them on the map.

To be able to model the appearance of unforeseen patients, the software allows at every step to *spawn* a patient on the map (manually or randomly generated), thus forcing the Planner to re-plan a solution.

The two Planner implementations are based on different approaches: the first relies on the Hungarian Algorithm for task assignment between agents, the second on the Particle Swarm Optimization algorithm and and heuristic to decode the plan. Both of them make use of the Maximum Expected Covering algorithm to find the optimal locations of idle ambulances.

Here follows an in-depth description of the implementation used.

Maximum Covering Location

Though the approaches mentioned in section 2 for solving the problem of *Maximum Covering Location* works in most of the cases, they do not take into account that all the areas must be covered by the facilities which proves to be pivotal in this case of placing ambulances. In many of the US municipalities, a patient demand needs to be met in less than 10 minutes 59 seconds [13] which may not be possible if the solution is skewed towards just the high demand locations and the distances are not taken into account.

The most recent development in this domain has been the use of a dynamic expected coverage model to determine the minimum number of ambulances and their locations for each time interval and solve it via a tabu search [14].

A domain specific approach has been used to solve the problem of placing a fixed number of ambulances in such a way that they optimally cover the city. The problem of placing the ambulances in such a configuration can be decoupled into a two pronged approach:

- Clustering the graph into K clusters so that each idle ambulance is in charge of one cluster.
- Finding the node in each cluster that minimizes the expected distance traveled by an ambulance, providing the demands of each node in a cluster.

The *Dijkstra* algorithm [15] which is a variant of the uniform cost search [16] is used to calculate the shortest path between every pair node in the map. These distances will then be exploited by the *Maximum Coverage Location* algorithm.

In short, the searching algorithm expands the shortest path from for each node until the goal node is reached. The algorithm can not cope with negative weights, but in our case this is not a limitation since the distance is never negative.

Faster algorithms exist to find all the shortest paths in a graph[17], but since the operation can be statically done when loading the map it doesn't affect the planning performances.

The input to the *Maximum Covering Location* algorithm is a graph and a demand for each node. The output is the preferred locations of the available ambulances ($N_{Available\_Ambulances}$) in such a way that the ambulances are able to cater to the demands of all the nodes and are also closer to the nodes with high demands.

To cluster the graph into K clusters, the *K-means* algorithm [18] uses the shortest distance between two nodes as a metric of similarity between a node and a cluster representative node.

Initially, the locations of the centroids of the clusters are chosen and each node is assigned to the centroid whose travel distance to that node is minimum. With n_k representing the *k-th* centroid:

$$nearestCentroid(n_j) = \underset{k \in 1 \dots K}{\operatorname{argmin}} \left[dist(n_k, n_j) \right] \;\; j \in 1, 2 \dots N \tag{1}$$

At each iteration, the centroid of each cluster is reassigned among the nodes of that cluster, so that the distance from the centroid to all the other nodes in the same cluster is minimum. Moreover, the distance from the potential centroid to every other node is weighted by the demand of the node. Formally:

$$centroid(C_k) = \underset{i \in 1 \dots N_{C_k}}{\operatorname{argmin}} \left[\sum_{j=1}^{N_{C_k}} dist(n_i^k, n_j^k) \cdot demand(n_j^k) \right] \;\; k \in 1, 2 \dots K \tag{2}$$

Where N_{C_k} is the number of nodes in the cluster C_k , n_i^k is the i^{th} node in cluster C_k, the metric $dist(n_i^k, n_j^k)$ denotes the shortest distance between the i^{th} node and the j^{th} node in cluster C_k , $demand(n_j^k)$ is the demand at the j^{th} node in cluster C_k.

The algorithm converges to the point that the position of the centroid inside every cluster remains the same, and consequentially the assignment of the nodes of the graph to a cluster. At this point, the centroid of each cluster represents the node that ensures the optimal coverage of the cluster.

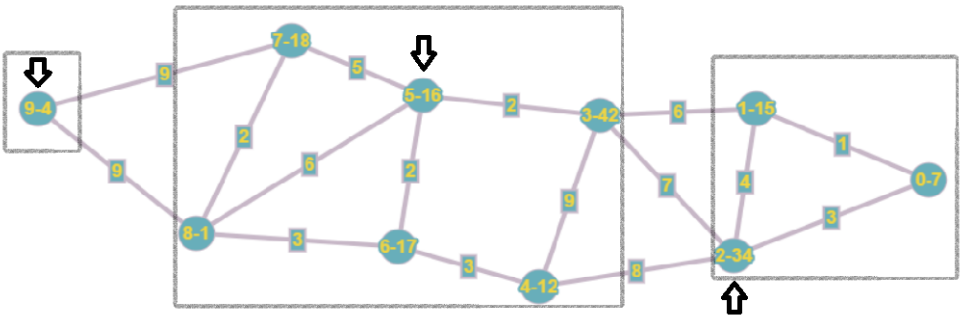


Figure 1: Max Coverage Algorithm with 10 nodes and 3 cluster. The second number inside each node is the demand. For each cluster the optimal location of an ambulance is pointed out.

Hungarian algorithm

One of the commonest ways to describe the assignment problem is the following. Consider a set of workers and a set of different tasks. To each worker performing a test is associated a cost. The goal is to assign tasks to workers under the following constraints:

- No more than one worker per task

- No more than one task assigned to a worker
- The total cost of the tasks is minimized

The problem can be formalized in matrix form. Given a two-dimensional matrix, choose one element in each row in a such way that each column has no more than one element chosen and sum of elements is minimized. Here follows an example where the optimal solution is highlighted.

$$\begin{pmatrix} 1 & ⑤ & 5 & 7 \\ ② & 6 & 8 & 13 \\ 8 & 12 & 1 & ⑤ \\ 14 & 6 & ② & 9 \end{pmatrix}$$

In our problem, the ambulances play the role of workers. The tasks can be seen as reaching a certain destination (i.e. a vantage point to serve the requests of a cluster) or a set of two waypoints (i.e. the patient first and then the nearest hospital). The distance covered by each ambulance represents the cost to minimize. Varying the number of ambulances available and the number of patients to serve different situations can occur:

$\#ambulances \geq \#patients$	The algorithm will send some ambulances to the patients and the spread the rest to optimally cover the city
$\#ambulances < \#patients$	The algorithm will send all the ambulances to serve a subset of the patients chosen with respect of their severity and distance

The output of the algorithm is the best distribution of tasks among ambulances. With the inclusion of the maximum coverage tasks it is also guaranteed that the number of tasks is at least the same as the number of ambulances, so each ambulance will be assigned with some location.

In order to solve the assignment problem in a polynomial time, the *Hungarian algorithm* [19] was implemented. Among several versions of the algorithm we decided to go with one that gives asymptotic $O(n^2m)$, where n is the number of workers and m is the number of tasks. The original idea of the implementation was proposed by Andrei Lopatin in 2008. A distinctive feature of the algorithm lies in its surprisingly briefness of code. The algorithm makes use of solving the maximal matching problem as its core idea.

Using this approach the following problem can occur in situations where the number of patients is greater than the number of vehicles.

$$\begin{pmatrix} 1 & ⑤ & 5 & 7 \\ ② & 6 & 8 & 13 \\ 14 & 6 & ② & 9 \end{pmatrix}$$

The following actions need to be taken:

Ambulance 0 goes to Patient 1, picks him up, drive to the hospital and deposits him

Ambulance 1 goes to Patient 0, picks him up, drive to the hospital and deposits him

Ambulance 2 goes to Patient 2, picks him up, drive to the hospital and deposits him

Clearly, the plan yielded does not offer a solution for the last patient. To overcome this problem we re-plan the solution as soon as an ambulance reaches the hospital thus being free again.

Note that a planning approach that is able to associate multiple patients to the same ambulance with no re-planning would yield a better plan. Consider as an example the situation in which an ambulance A0 is set into motion to save a patient P0, but as it gets closer to his position another ambulance A1 drops a patient P1 at an hospital and suddenly becomes the closest match to save P0.

Particle Swarm Optimization and Insertion Algorithm

Another approach to construct the plan is the *Particle Swarm Optimization* [20] (PSO) algorithm, often used as one of meta-heuristic methods for solving a vehicle routing problem.

PSO is not directly connected to a routing problem. It is used to find an optimal solution for a problem $\min f(x)$, where $f : R^n \to R$ is some function which is not required to be differentiable, contrary to several other optimization algorithms like gradient descent. Basically, no additional information about the function is required.

Initially, PSO constructs a random set of points in the sub-space of R^n. Each point is called a *particle*. Besides the position, it is also associated to a random velocity vector which represents the current movement of the particle. Each particle in a fixed position can be considered as a candidate for a problem solution, hence we can perform a search for a solution by iteratively moving particles in their current directions.

This search is guided: velocity vectors are not constant, they always change their direction to positions where better solutions were already found. More precisely, each particle has a memory of the best position that it has had at some iteration, and it constantly shifts to that location. Additionally, all particles know the overall best solution that has been reached among all particles, and also adjust their velocity vectors towards that position. Thus, for a particle p the update of its position on one iteration is calculated with the following formulas:

$$V_p \leftarrow \alpha V_p + \beta_l (X_p^{[best]} - X_p) + \beta_g (X_{global}^{[best]} - X_p) \tag{3}$$

$$X_p \leftarrow X_p + V_p \tag{4}$$

Where:

- V_p is the velocity vector of particle p;
- X_p is the position vector of particle p;
- α, β_l, β_g are coefficients;
- $X_p^{[best]}$ and $X_{global}^{[best]}$ are the best solutions found by the particle p and by all existing particles accordingly.

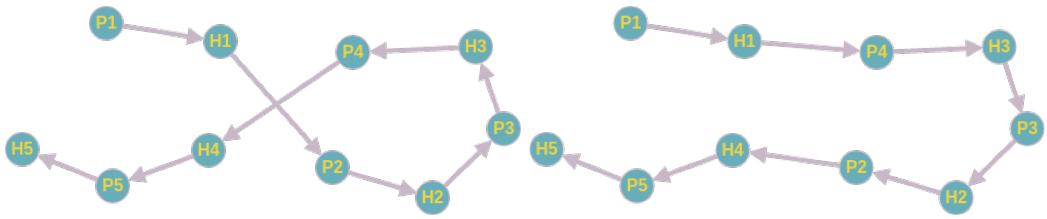
After a number of iterations the result is the best solution found by all particles, namely $X_{global}^{[best]}$.

Now the task is to apply PSO to a routing problem. We need to define f, the function to minimize in the routing problem. Here, we minimize sum of the distances that need to be covered by ambulances to pick and transfer all patients. Function f obtains not a plan which can be easily evaluated, but some vector from R^n space. Therefore, it is first needed to construct a plan from an n-dimensional vector. This is done using heuristic *Insertion algorithm*.

Let us consider a particle as a list of priorities for patients. That implies that the number of dimensions of a particles is equal to the number of patients waiting for an ambulance. These priorities are used to sort the array of patients - later patients will be added to the plan in the obtained order.

Insertion algorithm constructs the plan for all ambulances at once by adding patients to a plan one by one. On each iteration for a current patient it finds the best spot in a plan to insert it, so that insertion does not violate the rules (patients with higher severity are handled before those with lower severity) and the insertion cost is minimal. The insertion cost is the sum of traveling distances after a patient was added into the plan.

After all patients are inserted, a *2-opt* algorithm[21] is applied. For each ambulance it tries to reverse a part of the plan to decrease the plan cost, as it is shown on the figure. This allows to further optimize the constructed plan.



Plan for one ambulance Improved plan after 2-opt was applied

Figure 2: An example of applying 2-opt algorithm. Each graph denotes a version of a plan for the same ambulance, a plan is represented like: pick patient P_1, drop him at hospital H_1, pick next patient, ...

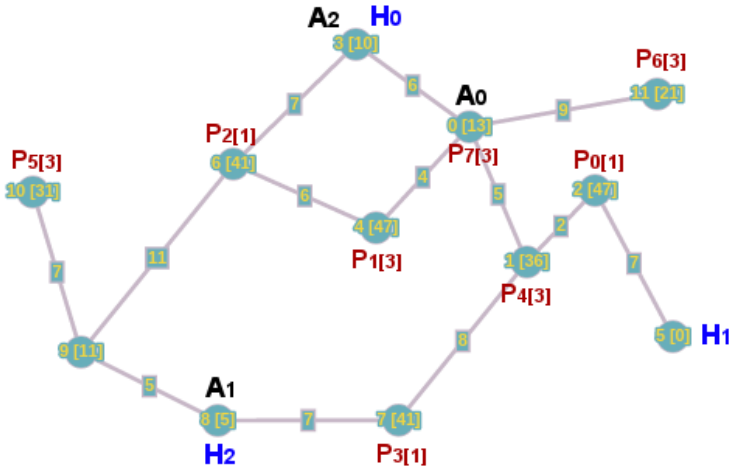


Figure 3: An example of city map with locations of patients, ambulances and hospitals. Demands are represented in for nodes while severities are the numbers in brackets for patients. Edge weights are rounded.

After the best plan to save the patients is obtained we consider if there are free ambulances and apply the *Maximum Expected Coverage* algorithm to dispatch them to the best location to cover the city.

3.2 Cases of study and experimental results

Since the PDDL problems are generated automatically using certain parameters, it is easy to change the problem factors in a way that allows the testing of different settings in the planner. That way we can explore the limits of our implementation and check which is the optimal set up of the algorithms for each problem.

Planning example

In order to test the correctness of planning algorithms, we ran them on several small city maps and checked the constructed plans manually.

Table 1 contains two plans built for the task represented on Figure 3. Both plans are valid: it can be noticed that patients P_0 and P_3 with the lowest severity (1) are always handled after all other patients with higher severity (3). Also, plans appear to be close to optimal. Hungarian Solver gives slightly worse result here, because it utilizes maximum coverage algorithm and also works better when new patients are constantly appearing, but this does not happen in this example.

Hungarian Planner	**PSO Planner**
$A_0 : P_7 \to H_0 \to P_6 \to H_0 \to P_2 \to H_0$ $A_1 : P_4 \to H_1 \to P_0 \to H_1 \to P_3 \to H_2$ $A_2 : P_1 \to H_0 \to P_5 \to H_2$	$A_0 : P_6 \to H_0 \to P_1 \to H_0 \to P_2 \to H_0$ $A_1 : P_5 \to H_2 \to P_3 \to H_2$ $A_2 : P_7 \to H_0 \to P_4 \to H_1 \to P_0 \to H_1$
Total distance traveled: 195.70 Total time patients waited: 48	Total distance traveled: 140.94 Total time patients waited: 47

Table 1: Constructed plans for Figure 3.

Map settings and performance: size and connection

Here we compare two planners by their running time. The model has too many parameters to compare algorithms by simply running them with various inputs. So, instead of doing it, we show the asymptotics for both algorithms, omitting some common parts like shortest distances pre-calculation.

In next formulas, $|N|$ is the number of nodes, $|P|$ is the number of patients, $|A|$ is the number of ambulances.

- Hungarian Planer:

$$O(|P| \cdot ([\text{Max Coverage algorithm}] + [\text{Hungarian algorithm}])) \tag{5}$$

$$O(|P| \cdot (\sum_{i=1}^{|A|} n_i^2 + |A|^2 \cdot |P|)) \tag{6}$$

$$O(|P| \cdot (|N|^2 + |A|^2 \cdot |P|)) \tag{7}$$

  In the formula 5, $|P|$ is present because the planner is rerun each time a patient is dropped in a hospital. Formula 6 uses n_i as the number of nodes in i^{th} cluster. In the worst case sum is equal to $O(|N|^2)$.

- PSO Planner:

$$O([\text{PSO planner}]) \tag{8}$$

$$O(\text{swarm\_size} \cdot \text{iter\_n} \cdot [\text{Particle decoding algorithm}]) \tag{9}$$

$$O(\text{swarm\_size} \cdot \text{iter\_n} \cdot ([\text{Insertion algorithm}] + [\text{2-opt algorithm}])) \tag{10}$$

$$O(\text{swarm\_size} \cdot \text{iter\_n} \cdot (|P| \cdot (|P| + |A|) + \sum_{i=1}^{|A|} p_i^2) \tag{11}$$

$$O(\text{swarm\_size} \cdot \text{iter\_n} \cdot (|P|^2 + |P| \cdot |A|) \tag{12}$$

  In the formula 11, the Insertion algorithm has asymptotic of $O(|P|^2)$, because for each patient it tries to insert it into the plan, and the amount of places to insert is equal to the number of ambulances plus number of patients already in the plan. Also, in the formula p_i denotes the number of patients in i^{th} ambulance's plan. It can easily be shown that the sum from the formula is not greater than $|P|^2$.

Results show that Hungarian planner is quite dependent on the number of nodes. However with a small number of patients, which is the best case for the algorithm, this is not a big problem since pre-calculation of all shortest distances also requires at least $N^2 \log N$ operations. In addition, Hungarian planner is more dependent on the number of ambulances and patients than PSO planner ($|P|^2|A|^2$ compared to $|P|^2 + |P||A|$). But at the same time the evaluation time of PSO planner is slowed by the size of a particle swarm and the chosen number of iterations.

Relative number of patients vs ambulances

To better grasp the limits of the algorithm we analyzed the planning achieved in the cases where the patients outnumber the ambulances and vice versa.

1. **#P <#A**: Having 100 nodes, 120 roads, demands have values in range $[0; 100]$, 10 ambulances, 10 hospitals, 5 initial patients and 100 more patients appear one by one after each step with probability 0.9.

   Results:

Hungarian Planner	**PSO Planner**
Total distance traveled: 36215 Total time patients waited: 426	Total distance traveled: 42891 Total time patients waited: 1235

   As it can be seen from the table, Hungarian Planner works much better than PSO Planner in this case. That probably happens because the former algorithm considers both patients and max coverage points at the same time, while the latter uses maximal coverage only after the main plan is constructed.

2. **#P >#A**: having 100 nodes, 120 roads, demands have values in range $[0; 100]$, 4 ambulances, 10 hospitals, 70 initial patients and 100 more patients appear one by one after each step with probability 1.

   Results:

Hungarian Planner	**PSO Planner**
Total distance traveled: 51769 Total time patients waited: 19461	Total distance traveled: 46783 Total time patients waited: 29237

   Here PSO Planner performs slightly better in terms of minimized distance. However, the total time patients have waited is still significantly lower. This is caused by the fact that PSO Planner minimizes the total traveling distance when choosing the best plan. The algorithm could be enhanced by replacing the PSO target function with one that considers patients' waiting times.

4 Conclusion

In light of the testing results, we can affirm that we solved different cases of study in two valid approaches. Moreover, these two methods are clearly distinguishable so that they can be used in real life to achieve legitimate needs.

Supposing a private company that values the fuel expenses, the need must account for saving overall distances. In the case of periodic patients, instead of emergencies, the chosen planner should be PSO, that minimizes the total distance when there are no uncertainties (the plan is fixed from the beginning to the end).

In the case of the Hungarian planner, as soon a an ambulance is free a new plan is calculated for it, while the rest of the ambulances move to high demand locations. The most probable location for a patient to appear will be at high demand nodes, so that the time waiting reduces.

On the other hand, the demand is a statistical measure. If a calamity hits a city, the probability of a patient appearing in high demand nodes will be completely different. In this case the PSO planner works better than the Hungarian planner because the appearance of patients would be completely random and the use of the Max Covering Location algorithm will not have a notable effect on the performance of the algorithms.

References

[1] M. Fox and D. Long, "Pddl2. 1: An extension to pddl for expressing temporal planning domains.," *J. Artif. Intell. Res.(JAIR)*, vol. 20, pp. 61–124, 2003.

[2] L. Talarico, F. Meisel, and K. Sörensen, "Ambulance routing for disaster response with patient groups," *Computers & Operations Research*, vol. 56, pp. 120–133, 2015.

[3] K. Knyazkov, I. Derevitsky, L. Mednikov, and A. Yakovlev, "Evaluation of dynamic ambulance routing for the transportation of patients with acute coronary syndrome in saint-petersburg," *Procedia Computer Science*, vol. 66, pp. 419–428, 2015.

[4] N. A. M. Nordin, N. Kadir, Z. A. Zaharudin, and N. A. Nordin, "An application of the a* algorithm on the ambulance routing," in *Humanities, Science and Engineering (CHUSER), 2011 IEEE Colloquium on*, pp. 855–859, IEEE, 2011.

[5] S. Panahi and M. Delavar, "Dynamic shortest path in ambulance routing based on gis.," *International journal of Geoinformatics*, vol. 5, no. 1, 2009.

[6] B. Baki, M. Bouzid, A. Ligeza, and A.-I. Mouaddib, "A centralized planning technique with temporal constraints and uncertainty for multi-agent systems," *Journal of Experimental and Theoretical Artificial Intelligence*, vol. 18, no. 3, pp. 331–364, 2006.

[7] R. Church and C. R. Velle, "The maximal covering location problem," *Papers in regional science*, vol. 32, no. 1, pp. 101–118, 1974.

[8] S. L. Hakimi, "Optimum locations of switching centers and the absolute centers and medians of a graph," *Operations research*, vol. 12, no. 3, pp. 450–459, 1964.

[9] E. Minieka, "The centers and medians of a graph," *Operations Research*, vol. 25, no. 4, pp. 641–650, 1977.

[10] J. E. Bell and P. R. McMullen, "Ant colony optimization techniques for the vehicle routing problem," *Advanced Engineering Informatics*, vol. 18, no. 1, pp. 41–48, 2004.

[11] Y. Nagata and S. Kobayashi, "A powerful genetic algorithm using edge assembly crossover for the traveling salesman problem," *INFORMS Journal on Computing*, vol. 25, no. 2, pp. 346–363, 2013.

[12] W. Hu, H. Liang, C. Peng, B. Du, and Q. Hu, "A hybrid chaos-particle swarm optimization algorithm for the vehicle routing problem with time window," *Entropy*, vol. 15, no. 4, pp. 1247–1270, 2013.

[13] J. B. Goldberg, "Operations research models for the deployment of emergency services vehicles," *EMS management Journal*, vol. 1, no. 1, pp. 20–39, 2004.

[14] H. K. Rajagopalan, C. Saydam, H. Setzler, E. Sharer, *et al.*, "Ambulance deployment and shift scheduling: An integrated approach," *Journal of Service Science and Management*, vol. 4, no. 01, p. 66, 2011.

[15] S. Skiena, "Dijkstras algorithm," *Implementing Discrete Mathematics: Combinatorics and Graph Theory with Mathematica, Reading, MA: Addison-Wesley*, pp. 225–227, 1990.

[16] S. J. Russell, P. Norvig, J. F. Canny, J. M. Malik, and D. D. Edwards, *Artificial intelligence: a modern approach*, vol. 2. Prentice hall Upper Saddle River, 2003.

[17] R. W. Floyd, "Algorithm 97: Shortest path," *Commun. ACM*, vol. 5, pp. 345–, June 1962.

[18] J. A. Hartigan and M. A. Wong, "Algorithm as 136: A k-means clustering algorithm," *Journal of the Royal Statistical Society. Series C (Applied Statistics)*, vol. 28, no. 1, pp. 100–108, 1979.

[19] H. W. Kuhn, "The hungarian method for the assignment problem," in *50 Years of Integer Programming 1958-2008*, pp. 29–47, Springer, 2010.

[20] J. Kennedy, "Particle swarm optimization," in *Encyclopedia of machine learning*, pp. 760–766, Springer, 2011.

[21] G. A. Croes, "A method for solving traveling-salesman problems," *Operations Research*, vol. 6, no. 6, pp. 791–812, 1958.

5 Appendixes

5.1 PDDL domain and problem definition

```
(define (domain ambulance world)
    (:predicates (Location ?l) (Road ?a ?b) (Patient ?p) (Ambulance ?a)
        (Hospital ?h) (At ?o ?l) (Available ?a) (Waiting ?p)
        (InHospital ?p) (In ?p ?a))
    (:functions (LocationCoord ?l) (LocationDemand ?l)
        (Distance ?a ?b) (Priority ?p))
    (:action move
      :parameters (?a ?from ?to)
      :precondition (and (Ambulance ?a) (Location ?from) (Location ?to)
        (At ?from) (Road ?from ?to))
      :effect (and (At ?to) (not (At ?from))))
    (:action pick
      :parameters (?p ?a ?l)
      :precondition (and (Patient ?p) (Ambulance a) (Location ?l)
        (Waiting ?p) (Available ?a) (At ?p ?l) (At ?a ?l))
      :effect (and (not (Waiting ?p)) (not (Available ?a)) (In ?p ?a)))
    (:action drop
      :parameters (?p ?a ?h ?l)
      :precondition (and (Patient ? p) (Ambulance ?a) (Hospital ?h)
        (Location ?l) (At ?a ?l) (At ?h ?l) (In ?p ?a))
      :effect (and (not (In ?p ?a)) (InHospital ?p) (Available ?a))))



(define (problem testcase1)
        (:domain ambulance world)
        (:objects l0 l1 l2 l3 l4 l5 l6 l7 l8 l9 p0 p1 p2 p3 p4 a0 a1 a2 h0 h1 )
        (:init (Location(l0))
               (= (LocationCoord(l0) 9 7))
               (= (LocationDemand(l0) 40))
               (Location(l1))
               (= (LocationCoord(l1) 4 2))
               (= (LocationDemand(l1) 38))
               (Road(l0 l1))
               (= (Distance(l0 l1) 7.485681917262148))
               (Location(l2))
               (= (LocationCoord(l2) 1 5))
               (= (LocationDemand(l2) 46))
               (Road(l1 l2))
               (= (Distance(l1 l2) 4.511922702038055))
               (Location(l3))
               (= (LocationCoord(l3) 2 2))
               (= (LocationDemand(l3) 31))
               (Road(l1 l3))
               (= (Distance(l1 l3) 2.7265334524534035))
               (Location(l4))
               (= (LocationCoord(l4) 2 8))
```

```
(= (LocationDemand(l4) 33))
(Road(l2 l4))
(= (Distance(l2 l4) 3.900457202840958))
(Location(l5))
(= (LocationCoord(l5) 7 7))
(= (LocationDemand(l5) 29))
(Road(l2 l5))
(= (Distance(l2 l5) 6.754512201134192))
(Location(l6))
(= (LocationCoord(l6) 8 3))
(= (LocationDemand(l6) 33))
(Road(l0 l6))
(= (Distance(l0 l6) 4.654738793870399))
(Location(l7))
(= (LocationCoord(l7) 7 1))
(= (LocationDemand(l7) 17))
(Road(l1 l7))
(= (Distance(l1 l7) 4.101798698353096))
(Location(l8))
(= (LocationCoord(l8) 2 9))
(= (LocationDemand(l8) 45))
(Road(l5 l8))
(= (Distance(l5 l8) 6.221093382749903))
(Location(l9))
(= (LocationCoord(l9) 2 0))
(= (LocationDemand(l9) 27))
(Road(l1 l9))
(= (Distance(l1 l9) 3.1281798458942482))
(Road(l2 l4))
(= (Distance(l2 l4) 3.561586615437787))
(Road(l0 l9))
(= (Distance(l0 l9) 10.710663181268329))
(Road(l6 l8))
(= (Distance(l6 l8) 8.776875399556682))
(Road(l0 l7))
(= (Distance(l0 l7) 7.25742037648318))
(Road(l1 l3))
(= (Distance(l1 l3) 2.9857429782113822))
(Road(l0 l8))
(= (Distance(l0 l8) 7.901308540492123))
(Patient(p0))
(= (Priority(p0) 1))
(Patient(p1))
(= (Priority(p1) 1))
(Patient(p2))
(= (Priority(p2) 3))
(Patient(p3))
(= (Priority(p3) 3))
(Patient(p4))
(= (Priority(p4) 3))
(Ambulance(a0))
```

```
        (Ambulance(a1))
        (Ambulance(a2))
        (Hospital(h0))
        (Hospital(h1))
        (At(p0 l2))
        (At(p1 l8))
        (At(p2 l0))
        (At(p3 l1))
        (At(p4 l4))
        (Waiting(p0))
        (Waiting(p1))
        (Waiting(p2))
        (Waiting(p3))
        (Waiting(p4))
        (At(a0 l7))
        (Available(a0))
        (At(a1 l7))
        (Available(a1))
        (At(a2 l6))
        (Available(a2))
        (At(h0 l6))
        (At(h1 l9)))
(:goal (InHospital(p0))
        (InHospital(p1))
        (InHospital(p2))
        (InHospital(p3))
        (InHospital(p4))))
```