An implementation of the Greedy Heuristic algorithms for the Maximal Blood Collection Problem

P. Ciambra *F. Usberti*

Relatório Técnico - IC-PFG-18-40
Projeto Final de Graduação
2018 - Dezembro

UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

An implementation of the Greedy Heuristic algorithms for the Maximal Blood Collection Problem

Pedro Ciambra Fábio Usberti*

Abstract

We provide implementations of the Greedy Heuristic and Greedy Recursive Heuristic proposed by Özener and Ekici (2018) for the Maximal Blood Collection Problem. In this problem, the objective is to to route blood donations from donation sites to a processing facility within a time limit for each donation.

1 Introduction

Routing algorithms have a broad horizon of applications, both in real-life logistic problems and as reductions to other, more abstract problems.

One family of real-world applications of routing algorithms are the Blood Collection Problem variants, defined by Özener and Ekici (2018), which consists of using a fleet of vehicles to route blood donations from their collection sites to a centralized processing center. The demand for such an algorithm is real and dire, considering that blood donations are necessary for many different medical procedures. In particular, those that necessitate platelets are worth mentioning; platelets have extremely short shelf life, and therefore need to be processed within a comparatively small time limit from collection time.

The US FDA (Food and Drink Administration) has regulated the collection of platelets accordingly, and determines that whole blood must be separated into its components preferably within 6 hours of collection time.

The difference to the Vehicle Routing Problem lies in this new limitation, which makes the problem more complex, as well as the accumulating nature of the donations in each collection site.

In literature, three variants of this problem are studied. The approach taken by Doerner et al. (2008) aims to minimize cost while collecting all available donations, while Yi and Scheller-Wolf (2003) try to minimize cost while collecting a pre-set amount of donations. The algorithms studied in this paper are based on a third approach, by Özener and Ekici (2018), which aims to maximize the amount of blood processed in total, given the opening and closing times of the donation sites and the availability of vehicles. They call this variant of the problem the Maximal Blood Collection Problem (MBCP).

*Instituto de Computação, Universidade Estadual de Campinas, 13081-970 Campinas, SP.

2 Definition of the MBCP

The MBCP can be defined as follows.

2.1 Variable definitions

Let the map of the geographical area that contains the collection sites be represented by an undirected, weighted, complete graph $G = (V, E)$, where v_0 represents the processing center and $v_1 ... v_N$ represent donation sites 1 through N. The weights of each edge correspond to the travel time between vertices, denoted as e_{ij} for the time between vertices i and j. One important remark is that, even if the original map of roads doesn't translate perfectly into a complete graph, one such can be obtained by computing the transitive closure of the incomplete graph, with no loss of generality.

Let the number of donations completed at donation site i and time t be denoted as d_t^i. This is not cumulative; it only pertains to the donations that became available *exactly* at time t.

Let the time limit to process a donation be K, and the number of available vehicles to be M. The vehicles have unlimited capacity, and start at the processing center.

Let t_α be the time of the first donation in the set (α its donation site), and t_ω be the time of the last donation (ω its donation site). The planning horizon, therefore, ranges from $t_\alpha - e_{0\alpha}$ to $t_\omega + \min(K, e_{\omega 0})$. We assume that the vehicles are always available, and that the processing center is always open.

Times are also treated as discrete numbers, both for the vehicles' motion and the times of the donations. In this implementation, we define the unit as one minute.

The original work also defines the opening and closing times of each donation center, but these are not used in the definition of the algorithm itself (therefore ignoring the case where the donation center closes before the donation can be picked up). Therefore, we opted not to define this information, as it can be inferred from the donation times.

These opening and closing times, however, are used in the random instance generator, which will be covered in section 5.2.

2.2 Rules for the vehicles

The motion of a vehicle can be defined by a (possibly empty) list of tours, where each tour starts and ends at the processing center and passes by a subset of the donation sites, picking up all available donations in each. Between each tour, a vehicle may wait for a period of time in the processing center.

The original work provides a proof that any solution that requires waiting at a donation site has an equivalent or better solution where all waiting happens at the processing center. One can easily convince oneself of this: by shifting any waiting spots to earlier in the tour, there is no possibility of losing any donations (the availability of the donations isn't a problem, since the times of the visits are being shifted to later and the donations will still be there. Spoiling is also not a issue, since the time of arrival at the processing center is ultimately the same).

Since this somewhat simplifies the problem, the proposed algorithms are designed to take this into account. Therefore, in this model, we consider that each stop in the donation sites happens instantly.

In this model, each vehicle may:

- traverse an edge of the graph,
- pick up available donations in a donation site,
- drop off donations at the processing center,
- wait for an arbitrary amount of time at the processing center.

Only the graph traversal and the waiting actions consume time; we assume the others are executed instantly.

3 Strategies considered in the literature

The original paper considers a number of approaches to the problem. One of them is the integer programming approach, where a mathematical model is formed from the problem definition and the variables and restrictions are fed into a linear solver. This approach's performance is found not to scale well beyond very small instances.

Afterwards, the authors attempt to simplify the problem by breaking it into parts. Also, a new restriction is introduced: each donation site is now assigned to a vehicle, and can only be visited by that vehicle. It also defines an upper limit to the number of donation sites a single vehicle may serve. This introduces the problem of grouping the donation sites into clusters, but also simplifies the routing part of the problem, since now there is only one vehicle for each cluster.

For each part of this breakdown, the original paper provides pseudo-code describing their approach. For the actual routing part (with a single vehicle and limited donation sites), they provide and evaluate four different algorithms. In this paper, we attempt to evaluate two of them: Greedy Heuristic and Greedy Recursive Heuristic.

4 Algorithm definitions

The authors of the original paper suggest an approach that interweaves the node clustering and route calculation algorithms, alternating the output of one to the input of the next. This results in an iterative, bottom-up approach that attempts to grow the clusters from a collection of seeds (for whose calculation Algorithm 1 is provided). This interweaving process is laid out in Algorithm 2, and takes a single-vehicle routing strategy as a parameter.

As mentioned, four different approaches for this single-vehicle routing are provided: GH, GRH, PGH and IPGH. We decided to implement and measure GRH (the one that offered the best measured results); and, since it requires GH as a subroutine, we have also measured its performance for no extra effort.

The pseudo-codes for these algorithms (1 through 4) are provided in the original paper, but the implementation details and choices are not specified by the authors.

4.1 Seed generation

The seed generation (Algorithm 1) is straightforward. It operates like the GH and GRH algorithms, in the sense that it slides a window from the end of the time horizon to the beginning, counting the collected donations for each site as if just arriving at the processing center. Once a point in time is found where enough sites have a positive sum of collected donations, we choose the sites with the largest sums as our seeds.

This algorithm takes a λ parameter for the sliding window.

4.2 Clustering

This algorithm uses the single-vehicle sub-problem solutions to determine to which cluster assign each donation site. It does this by calculating a marginal benefit for each cluster; it runs the single-vehicle strategy for each cluster independently, and then with the addition of the node; the cluster that benefits the most from the addition of the node is chosen and immediately added to the cluster, before going on to the next site to consider.

In the end, when all nodes have been assigned to a cluster, the single-vehicle algorithms may be run again to determine the actual paths.

4.3 Greedy Heuristic

This algorithm (GH, listed as Algorithm 3 in the original paper), as the name implies, approaches the problem with a greedy strategy. One particularity is that it advances backwards in time, setting the time of the last arrival of the vehicle in the processing center and retroactively considering the best choices of nodes to construct a path, taking into account the donations found in each donation center. Each chosen donation is marked as taken, so that it won't be considered in subsequent passes, and the process continues until there are no more donations to be collected.

This algorithm also takes a λ parameter, which dictates the amount of time the vehicle waits at the processing center when there are no reachable donations at that point in time. The lower this parameter, the most precise the algorithm, at the cost of higher computational times.

4.4 Greedy Recursive Heuristic

This algorithm (GRH, is listed as Algorithm 4 in the original paper) is very similar to GH, only differing in the method of choosing the previous node. Despite the name, it is not recursive; instead, it uses GH as a subroutine. While GH only goes for whichever node provides the greatest amount of redeemable donations at that point in time, GRH actually runs a modified version of GH for every node choice, in every round, in an attempt to approximate the total number of donations that each node choice will result in for the whole time horizon.

Unsurprisingly, this algorithm is significantly slower than the previous one. The same λ parameter from GH applies here, however it's possible to attribute a different parameter for the outer GRH algorithm and the inner GH that happens multiple times for every step of the route.

5 Implementation

It's noteworthy that the original paper does not provide details of the specifics of the algorithm implementations, other than mentioning the technology used (C++ and CPLEX Concert Technology). As such, we have made implementation choices that we believe will optimize the calculation speeds for the specified instance sizes.

Our implementation uses only C++11 and the standard library. No external libraries were required.

5.1 Representation choices

All mentioned algorithms utilize abstractions such as sets and graphs.

For the sets, we chose to use bitset representation (where each bit of a variable represents the presence of an entity in the set). For the clusters and vehicles, we use 32-bit integers (since, conveniently, the maximum amount of donation sites is 30), and for the donations in each site, we use 64-bit integers (since each site may have at most 50 donations). This is useful, because set operations can be executed with very few processor instructions, and it's the most compact representation conceivable, saving memory use (and therefore, decreasing the amount of cache misses).

For the weighted graph, we use adjacency lists (stored in a fixed-size, memory-contiguous C array). This is done to make sure that all necessary memory is allocated before the algorithm runs and that no time will be lost with time-expensive memory allocation during processing. Being memory-contiguous also decreases the frequency of cache misses.

The donation times are also stored in memory-contiguous C arrays, sorted by collection time. However, the implemented algorithms do not take advantage of this sorting. The algorithm could be changed to make use of this, as could some other ideas like using segment trees instead of plain arrays; however, the small size of the arrays make this a likely inconsequential optimization.

All instance data is stored in a memory-contiguous structure, containing the adjacency list, the donation time lists, and the times of the first site's opening and last site's closing (which are necessary for the algorithms).

5.2 Instance generation

The instance generation follows as closely as possible the rules defined in the original paper, as best as our interpretation could allow (the lack of source code meant we had to make some assumptions).

The graph is generated by spreading 31 points across a 2D space and separating them in two groups: one uniformly distributed across the whole map and one using normal distribution around 4 clusters of nodes (to simulate urban environments with dense population).

One of the points is chosen at random to be the processing center, and the distances between points are calculated as the euclidean distance between each position. The original positions of each node in the 2D plane are not stored. Afterwards, the opening and closing times of the nodes are generated, and using that information, the donations for each node (also using a mixture of uniform and normal distribution). Only the first opening time and last closing time are stored, as the remaining information is not used by the algorithms.

All information is stored as signed integers.

5.3 Seed generation

For the normal distribution, we use the standard library's normal distribution implementation. For random numbers, we simply use `rand()`.

The specific positions of each "urban area" are not specified; we have decided to fix them at four corners of a 500x500 square, centered on the 1000x1000 map.

5.4 Clustering algorithm

The original paper doesn't specify the behavior when no cluster has a positive marginal benefit, or in the case of a tie; we have decided to, in these cases, attribute the node to the cluster that suffers the least from the addition of the node, and to resolve ties by choosing randomly from the winners.

5.5 Single-vehicle routing algorithms

Both algorithms are very similar in implementation. We decided to pass two extra parameters, LastNode and LastTime, so that the same GH implementation can be used on its own or as the GRH subroutine. We also provide an optional parameter to store the generated path, but we do not use it in our implementation (which is focused on the performance test). It is there for the user's convenience, however.

The marking of taken donations is also done using bitsets. Furthermore, we discard the keeping of the EndTime return variable, since it can be trivially recovered from the paths list.

6 Measurements

The original work compares each algorithm by its performance in comparison to a upper bound, defined by relaxing the constraint of the number of vehicles and allowing each donation site to have its own vehicle. This upper bound is calculated by finding the maximal weight independent set in a interval graph defined by the arrival times in the site. Again, the paper does not provide the implementation, which leaves us to implement it ourselves.

For each donation site, we generate an interval array of 800 integers, each corresponding to the weight of the interval node (that being the amount of donations that can be collected if arriving at the node at that point in time). We do not need to actually store the edges, since the connection is defined by a mathematical formula (two nodes are connected if the difference of their arrival times is less than twice the distance between the node and the processing center). The proof that this works is included in the appendix of the original paper.

We then calculate the maximum weighted independent set, and take the result as the maximum amount of donations that can be collected in that site. The sum of this result for all sites is the upper bound that will be used as a reference to measure the performance of the algorithms.

We calculate the maximum weighted independent set with a simple linear dynamic programming algorithm, where we decide whether to include a time for a visit or not. For each time t in the planning horizon, we define $dp[t]$ as the maximum number of donations collectable by visiting the site at most at t. Therefore, $dp[0] = weight[0]$, and, for each subsequent time, $dp[t] = max(dp[t-1], weight[t] + dp[r])$, where $r = dist(site, 0)$.

It's worth noting that this algorithm is linear with the planning horizon, which makes it very fast to compute in comparison with the other algorithms, given the fact that T is a relatively small integer.

To compare with the original work, we have set the parameters to 7 vehicles, 300 minutes of processing time limit and maximum cluster size = 8.

7 Results

We have obtained very different results from the original paper.

For starters, our versions of the algorithms take less than one second to run (for a single dataset), as opposed to the several minutes reported by the original paper. This might be explained by the original work's reliance on a heavy framework with a large overhead, instead of pure C++. However, the measured solution quality (effectiveness in comparison to the upper bound) is a lot lower: 52.707% for GH, and 51.376% for GRH (as opposed to 76% and 78%, respectively). This might be explained by differences in implementation details for each part of the project, from the instance generation to the way the upper bound is calculated.

One point worth mentioning is that, in our version, GRH seems to perform consistently worse than GH.

8 Suggestions for future work

8.1 Views on the proposed algorithms

In this section, we provide our views on some of the studied algorithms.

8.1.1 Seed generation

This algorithm has some issues; by enforcing the selected nodes to be plentiful at the same or close to the same time period, without checking that they are also far apart, we risk fragmenting a cluster of sites that are physically close together, and that could be otherwise successfully served by a smaller number of vehicles (freeing the others to collect more donations from further away).

8.1.2 Clustering

In this algorithm, the order in which the sites are processed matters, since subsequent sites will be matched against bigger clusters than previous ones. Since the order of the nodes is arbitrary, this is a flaw of this algorithm; we believe it would be more fair if each node was compared against each cluster in parallel, and then assigned to the most beneficial one (with some policy to resolve ties).

8.1.3 Upper bound calculation

The original work provides only the final results of the upper bound calculation, but does not specify a detailed algorithm. Instead, the authors supply a reference to a research paper (Hsiao et al. 1992) which proposes an algorithm for calculating the maximum weighted independent set in a interval graph in O(n log n) time. We have not replicated this algorithm in this paper, instead opting to use a simpler, dynamic programming approach that takes linear time with respect to the planning horizon instead of considering the number of donations (which is facilitated by the small size of the instances). We suggest that the impact and accuracy of different ways to generate an upper bound be studied in future work, in particular to optimal solutions obtained from exact algorithms based on branch-and-bound and branch-and-cut techniques.

9 Conclusions

With this work, we conclude that, in spite of the complexity of the problem, significant improvement can be done to the execution times by using simple low-level optimization techniques; our implementation had running times orders of magnitude smaller than the ones reported in the original work. This also implies that it could be feasible to run even the slowest of the algorithms from the original paper with even larger instance sizes (maybe even the integer linear programming solution). We leave this to be attempted in future work.

Furthermore, because of the absence of the original implementations and instances, any qualitative comparison is compromised. There were several implementation details that we have assumed or guessed, to which we attribute the relative differences in the obtained results.

This is a case of study where there may be much room for improvement, which would be greatly welcome as this has direct impact in the efficiency of health institutions.

All code for our implementation can be found at *https://gitlab.com/jabcross/blood-collection-problem.*

10 Bibliography

References

[1] Okan Örsan Özener, Ali Ekici *Managing platelet supply through improved routing of blood collection vehicles*, Computers and Operations Research (2018)

[2] Doerner et al. *Exact and heuristic algorithms for the vehicle routing problem with multiple interdependent time windows*, Computers and Operations Research (2008)

[3] Jinxin Yi, Allan Scheller-Wolf *Vehicle Routing with Time Windows and Time-Dependent Rewards: A Problem from the American Red Cross*, Manufacturing Service Operations Management (2003)

[4] Hsiao et al. *An efficient algorithm for finding a maximum weight 2-independent set on interval graphs* Information Processing Letters (1992)