A Greedy Randomized Adaptive Search Procedure (GRASP) for the multi-vehicle prize collecting arc routing for connectivity problem

A Greedy Randomized Adaptive Search Procedure (GRASP) for the multi-vehicle prize collecting arc routing for connectivity problem

Luana Souza Almeida[a*], Floris Goerlandt[a], Ronald Pelot[a], Kenneth Sörensen[b]

a Dalhousie University, Department of Industrial Engineering, Maritime Risk and Safety (MARS) Group, 5269 Morris Street, Halifax, Nova Scotia, Canada, B3H 4R2

b University of Antwerp, Department of Engineering Management, ANT/OR - Operations Research Group, Prinsstraat 13, 2000 Antwerp, Belgium

* Corresponding author: luana.almeida@dal.ca

Declarations of interest

The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: Floris Goerlandt reports financial support was provided by Marine Environmental Observation Prediction and Response Network. Ronald Pelot reports was provided by Marine Environmental Observation Prediction and Response Network. Floris Goerlandt reports financial support was provided by Province of British Columbia. Ronald Pelot reports financial support was provided by Province of British Columbia. Ronald Pelot reports a relationship with Marine Environmental Observation Prediction and Response Network that includes: board membership, funding grants, and travel reimbursement.

Abstract

Natural disasters such as earthquakes can severely impact road networks. Depending on the disaster intensity and the size of the affected area, the network may be divided into multiple disconnected parts. In a disaster response context, decision-makers need to determine the roads that should be unblocked to facilitate relief activities such as search and rescue, evacuation, and distribution of emergency supplies. The multi-vehicle prize collecting arc routing for connectivity problem (KPC-ARCP) is a well-known problem dealing with such a scenario. A matheuristic to solve the KPC-ARCP was proposed in previous research, which tested instances with fewer than 400 vertices and 700 edges. However, it is unknown whether the matheuristic can handle larger instances. This article proposes a Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic with the hypothesis that GRASP is faster and can solve more extensive networks. Two sets of tests are performed on randomly generated instances with increasing size. The gap in the objective function values and the execution times of GRASP versus the matheuristic are compared. The results indicate that GRASP can achieve objective function values as good as the matheuristic and is significantly faster depending on the parameter settings.

Keywords

Metaheuristics, natural hazard, network connectivity, humanitarian logistics, emergency response

1 Introduction

Road networks are susceptible to be damaged or blocked after natural disasters, especially earthquakes. For instance, Norton (2018) and the National Police Agency of Japan (2020) state that after the 2011 earthquake and tsunami in Japan, the government had to manage approximately 27 million tons of debris and restore around 115 bridges and 4,200 roads. Norton (2018) also asserts that reinstating the infrastructure in the Tohoku region took nearly two years. Depending on the intensity and scale of the event, vulnerable communities may become unreachable by land. Masson (2014) investigates the impacts of the 2010 Hurricane Igor in Atlantic Canada and highlights that damaged roads and bridges isolated communities in the Burin and Bonavista peninsula for eleven days.

During the disaster's immediate and sustained response phase, isolated communities need to be reconnected by land to facilitate emergency activities such as evacuation, search and rescue, and especially the distribution of relief supplies. As an illustration, Sato & Suzuki (2013) state that the tsunami that followed the Great East Japan Earthquake wrecked the food inventories in the coastal areas of Iwate Prefecture. The region became vulnerable and dependent on emergency goods. Previous studies proposed models that attempted to minimize the completion time of road unblocking activities. However, affected areas and communities can have different resilience levels (Bruneau et al., 2003). Therefore, prioritization of which roads to unblock may be important. To this effect, Akbari & Salman (2017a) propose the multi-vehicle prize collecting arc routing for connectivity problem (KPC-ARCP), where K stands for multiple vehicles. In this problem, distinct priorities are attached to the disconnected parts of the system.

Concerning the solution method applied to large-scale KPC-ARCP, Akbari & Salman (2017a) propose a matheuristic and test its performance on different instances. Their largest data set has 349 vertices and 689 edges. Nevertheless, depending on the disaster's severity and scale, such a maximum network size may not be realistic in actual disaster scenarios. For example, Tatham & Kovács (2007) find that the 2005 Pakistan Earthquake affected an area with a radius of 100km from its epicentre, which corresponds roughly to the size of Belgium. Sakuraba et al. (2016) show that the road network for the 2010 earthquake in Port-au-Prince (Haiti) has 16,657 vertices and 19,558 edges. Similarly, Yan & Shih (2012) employ their model on a real case network of the 1999 Chi-Chi earthquake in Taiwan, which contains 15,845 vertices and 65,460 edges. Thus, it is unlikely that in realistic conditions, a network will have a size as small as the largest instance tested for the KPC-ARPC by Akbari & Salman (2017a). Even though these authors demonstrated that the KPC-ARCP matheuristic is efficient on instances up to 400 vertices, there currently has been no investigation to evaluate if it can handle more extensive networks. Their approach consists of a hybrid between heuristics and a Mixed Integer Programming (MIP) model. For larger network sizes, solving the MIP model may pose a challenge. Consequently, it may be impracticable to use the matheuristic on large networks.

This paper introduces a Greedy Randomized Adaptive Search Procedure (GRASP) metaheuristic to solve the KPC-ARCP. We consider the hypothesis that GRASP can solve larger instances than matheuristic in less time. To investigate this, the two solution methods are tested on randomly generated data sets with

increasing sizes. We compare the objective function values and execution times to determine which solution method performs better on large-scale networks.

The structure of this paper is as follows. First, a brief description of KPC-ARCP is given in Section 2. Then the relevant literature on road clearing problems is briefly reviewed in Section 3. Section 4 describes the methodological approach, while Section 5 presents the results. Section 6 discusses the limitations of the proposed method and suggests future research directions. Section 7 concludes.

2 Problem Definition

This section briefly describes KPC-ARCP. The authors recommend Akbari & Salman (2017a) for an elaborated description and Sections 4.1.2 and 4.1.3 for further details.

Consider an undirected graph represented by $G = (V, E)$ where V is the set of vertices and E is the set of edges. The set of vertices contains the affected communities and the roads' intersections, whereas the set of edges includes intact and blocked roads. Each edge has a traversal time, and if it is blocked, there is an additional cost representing the expected time to clear that edge. Figure 1 depicts the scenario of the disaster. First, the initial state of the network when all the roads are intact is represented in Figure 1 (a). The blocked roads are displayed in Figure 1 (b) with dashed lines. Note that if the subset of blocked edges is removed from the graph, as they are not traversable, the resulting graph is disconnected, as in Figure 1 (c). Each group of connected vertices in the disconnected graph is called a component. For instance, in Figure 1 (c), vertices 2, 3 and 6 represent a component. Components have an associated "prize" that drives the optimization process, which can for example be a priority weight or the size of the isolated population. A summary of KPC-ARCP's sets and parameters is available in Appendix A.

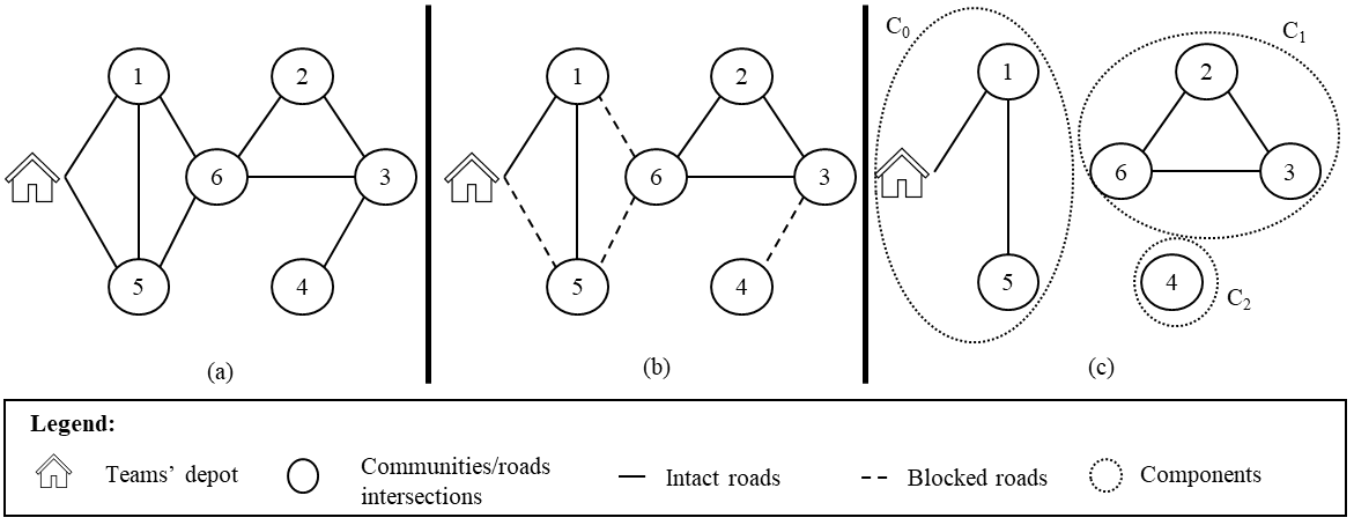


Figure 1 - Representation of the road clearing scenario (a) before the disaster, (b) after the disaster and (c) isolated areas. Based on Akbari and Salman (2017a)

A road clearing team represents the required machinery, equipment, and workforce. In the KPC-ARCP, all teams start their route on the same vertex (depot), which in this paper is assumed to be the vertex with index zero. The teams do not return to their depot after each shift because it is considered that there are

enough resources to work continuously. The duration of the team's route must be shorter than a pre-determined time limit (e.g. 72 hours).

KPC-ARCP considers the synchronization of the teams. First, a blocked edge is only traversed if it was previously unblocked (i.e. it is not possible to bypass a blocked edge). Secondly, the first team to arrive at one end of a blocked edge is commissioned to clear it. Thirdly, if another team is unblocking the edge, then the others must wait to cross it, because it is assumed that only one team is allowed to clear the road. In addition, an unblocked edge is available in both directions, and blocking times are considered only in the first traversing turn. In this paper, an unblocked road is also denoted as a traversable road.

Finally, the objective of KPC-ARCP is to maximize the total prize collected within a pre-determined time interval. Thereby, the expected outputs of this problem are the objective function value, the list of edges to be cleared, and the route of each road clearing team.

3 Literature review

This section reviews the relevant literature related to the multi-vehicle prize collecting arc routing for connectivity problem (KPC-ARCP). The focus is on the main characteristics and solution methods of quantitative road clearing models, which are summarized in Table 1. The authors recommend Çelik et al. (2016) for a detailed review on network restoration.

One type of approach to the road clearing problem determines a subset of roads that should be repaired without defining the teams' routes or schedule. Maya Duque & Sörensen (2011) study the repair and upgrade of the roads that connect towns to rural centres. They propose a model with a non-linear objective function that attempts to maximize accessibility while considering a tight budget and a limited workforce. The authors propose a Greedy Randomized Adaptive Search Procedure (GRASP) combined with Variable Neighbourhood Search (VNS) as a solution method. In contrast, in this paper, we consider the routing and scheduling of multiple repair teams.

Some authors study the routing and scheduling of the road clearing teams in the context of accessibility improvement. For instance, Maya Duque et al. (2016) present the Network Repair Crew Scheduling and Routing Problem (NRCSRP). A single repair crew starts from a depot and has to optimize the accessibility to vulnerable communities in the shortest time. Moreno et al. (2019) and Moreno et al. (2020b) propose alternative solution methods to NRCSRP. Kim et al. (2018) expand NRCSRP by considering that repairing times may change over the time horizon due to earthquake aftershocks. In Moreno et al. (2020a), multiple repair crews with different expertise can work only on specific road damage types. KPC-ARCP is different because the focus is to reconnect the network based on the areas' priorities instead of accessibility improvement in the shortest time.

Sahin et al. (2016), Berktaş et al. (2016), and Ajam et al. (2019) consider the problem of a single team that repairs the roads and distributes relief supplies. Sahin et al. (2016) present the Debris Removal in the Response Phase Problem (DRP). An emergency crew must visit a pre-determined set of critical facilities (e.g. shelters, hospitals, and schools) and unblock roads only if necessary. Berktaş et al. (2016)

build up on DRP. Besides a new mathematical formulation, they propose the prioritized DRP so that critical facilities have different priorities to be visited. Ajam et al. (2019) introduce a matheuristic and a metaheuristic to the DRP. These problems differ from KPC-ARCP as there is only one team, and the set of critical facilities to be visited is known beforehand.

Problems with multiple clearing teams are addressed in the literature by Tuzun Aksu & Ozdamar (2014), Sakuraba et al. (2016), Yan & Shih (2007), Yan & Shih (2012), and Vodák et al. (2018). Tuzun Aksu & Ozdamar (2014) tackle the creation of a restoration schedule without considering the teams' routes. Their model minimizes the completion time and assumes that roads have distinct priorities. Sakuraba et al. (2016) describe the Road Emergency Rehabilitation Problem (RERP) consisting of two sub-problems: determination of the shortest path to refugee centres and the repair schedule of blocked arcs. Barbalho et al. (2020) propose a GRASP and an Iterated Local Search (ILS) to the multi-period Work-troops Scheduling Problem (WSP). The objective of WSP is to improve the accessibility of a network by scheduling multiple teams to clean roads over a time horizon. The problem addressed by Sakuraba et al. (2016) and Barbalho et al. (2020) differs from KPC-ARCP because they focus on improving the accessibility in a connected graph.

In the work by Yan & Shih (2007), the goal is to minimize the required time for emergency repairs in a time-space network. Yan & Shih (2012) expand this problem by proposing an alternative solution method that uses an Ant Colony System based Hybrid algorithm (ACSB). The authors address the routing and scheduling of teams that start at various depots. The objective is to visit all repair points in the shortest time. In contrast, KPC-ARCP considers only one depot and the problem is designed to reconnect the network instead of visiting all repair points.

Similar to Yan & Shih (2012), Vodák et al. (2018) implement an Ant Colony Optimization (ACO) metaheuristic. Their model aims to minimize the total time required by multiple teams to reconnect a network's components. However, the teams are allowed to traverse blocked roads, and thus it is not necessary to synchronize the routes. Unlike Vodák et al. (2018), KPC-ARCP does not allow roads to be bypassed, and the components have different priorities.

Another perspective in the road clearing literature assumes that the network will be fragmented into multiple components after the disaster. In other words, the initial state of the network is disconnected. Then, the focus is to recover the connectivity between components. Some authors that deal with such a network condition are Asaly & Salman (2014), Kasaei & Salman (2016), Akbari & Salman (2017a), and Akbari & Salman (2017b).

Asaly & Salman (2014) are the first to study the Arc Routing for Connectivity Problem (ARCP). In their case, there is a single crew leaving a single depot to reconnect a directed graph. In their approach, blocked roads are not traversable unless they are cleared. Kasaei & Salman (2016) address the undirected version of ARCP. Akbari & Salman (2017b) study the case of multiple crews and multiple start points and name the problem as K-ARCP, where K stands for multiple teams. Akbari et al. (2021a) analyze K-ARCP further and suggest an alternative solution method called Mixed Integer Programming based heuristic

with Rich Local Search (MIP-RLS). According to Akbari et al. (2021a), the MIP-RLS has a better performance than the matheuristic proposed by Akbari & Salman (2017b). In the MIP-RLS, an initial solution is generated by pre-processing the problem and solving a binary program. Then, the initial solution is improved with rich neighbourhood moves. These problems focus on minimizing the time to reconnect the network, which is not the objective of KPC-ARCP.

Akbari et al. (2021b) propose the Online Road Clearance for Connectivity Problem (ORCCP), an extension of the ARCP. In ORCCP, the unblocking times of the damaged roads are unknown. When a team arrives at an end-node of a blocked road, the unblocking time is revealed online. The authors propose two solution methods: a MIP approach and a polynomial-time online algorithm, also known as the Thresholding algorithm. KPC-ARCP differs from Akbari et al. (2021b) as the unblocking times are assumed to be known in advance, and there are multiple road clearing teams.

The prize collecting version of ARCP is studied in Kasaei & Salman (2016). A single repair crew leaves the depot to reconnect a network's components in a delimited time range. In this problem, each affected area has a different priority to be reconnected. The authors show that PC-ARCP is NP-Hard. Then, Akbari & Salman (2017a) extend PC-ARCP to the case with synchronized multiple repair crews and a single depot. This variant is known as the multi-vehicle prize collecting arc routing for connectivity problem (KPC-ARCP), and it is NP-Hard. Akbari & Salman (2017a) propose a Mixed Integer Programming model (MIP) and a matheuristic to solve large instances. The matheuristic was tested on random data sets and in three sets of instances in Istanbul, with the largest instance having 349 vertices and 689 edges.

4 Methodology

The purpose of this paper is to compare the performance of GRASP and the matheuristic on KPC-ARCP. The structure of the overall methodological approach is shown in Figure 2. The present section starts by explaining the proposed algorithm of GRASP to KPC-ARCP in Section 4.1. Next, the procedure to create the set of instances is described in Section 4.2. Finally, the two tests used to identify the GRASP parameters and the ones to compare GRASP with matheuristic are explained in Section 4.3. Details concerning the specifics of Figure 2 are given in the subsequent sections.

The authors refer to Akbari & Salman (2017a) for a detailed explanation of the matheuristic. The matheuristic to solve large KPC-ARCP instances is named Algorithm 2 – Successive Single Vehicle Algorithms (SSV). Akbari & Salman (2017a) state that the complexity of the exact model for KPC-ARCP comes from the teams' synchronization constraints. Therefore, solving the single vehicle case is easier and faster because there is no timing conflict. The exact model to the single vehicle model is denoted as PC-ARCP-II-(P^k) where P^k stands for the prize set of team k. The matheuristic works as follows. For each one of the teams, solve the PC-ARCP-II-(P^k) and update the prizes of the visited components to zero. The update on the prizes' values prevents the connection of the same components by multiple teams.

1 **Table 1 - Literature review of road clearing models**

Reference	Focus on recovering network connectivity	Teams	Synchronized teams	Objective function	Solution Method
Maya Duque & Sörensen (2011)	No	Multiple	No	Accessibility time	GRASP, VNS
Sahin et al. (2016)	No	Single	No	Completion time	Mathematical model, heuristic
Berktaş et al. (2016)*	No	Single	No	Completion time / Prize collection	Mathematical model, heuristic
Ajam et al. (2019)	No	Single	No	Completion time	Matheuristic, GRASP, VNS
Maya Duque et al. (2016)	No	Single	No	Accessibility time	Dynamic Programming, GRASP
Moreno et al. (2019)	No	Single	No	Accessibility time	Brand-and-Benders-cut (BBC)
Moreno et al. (2020b)	No	Single	No	Accessibility time	Genetic Algorithm, Simulated Annealing, Hybrid approach (metaheuristic + BBC)
Kim et al. (2018)	No	Single	No	Accessibility time	Ant Colony System
Moreno et al. (2020a)	No	Multiple	Yes	Accessibility time	Mathematical model
Yan & Shih (2007)	No	Multiple	No	Completion time	Mathematical model, heuristic
Yan & Shih (2012)	No	Multiple	No	Completion time	Ant Colony System based Hybrid algorithm (ACSB)
Tuzun Aksu & Ozdamar (2014)	No	Multiple	No	Accessibility time	Mathematical model
Sakuraba et al. (2016)	No	Multiple	Yes	Accessibility time	Mathematical model, heuristic
Barbalho et al. (2020)	No	Multiple	Yes	Accessibility time	GRASP, ILS
Asaly & Salman (2014)	Yes	Single	No	Completion time	Mathematical model
Kasaei & Salman (2016)	Yes	Single	No	Completion time / Prize collection	Mathematical model, heuristic
Akbari & Salman (2017b)	Yes	Multiple	Yes	Completion time	Mathematical model, matheuristic
Vodák et al. (2018)	Yes	Multiple	No	Completion time	Ant Colony Optimization
Akbari et al. (2021a)	Yes	Multiple	Yes	Completion time	Mixed Integer Programming-based heuristic with Rich Local Search
Akbari et al. (2021b)	Yes	Single	No	Completion time	Mathematical model, Thresholding algorithm
Akbari & Salman (2017a)	Yes	Multiple	Yes	Prize collection	Mathematical model, matheuristic
This paper	Yes	Multiple	Yes	Prize collection	GRASP

2 * Papers that present more than one model

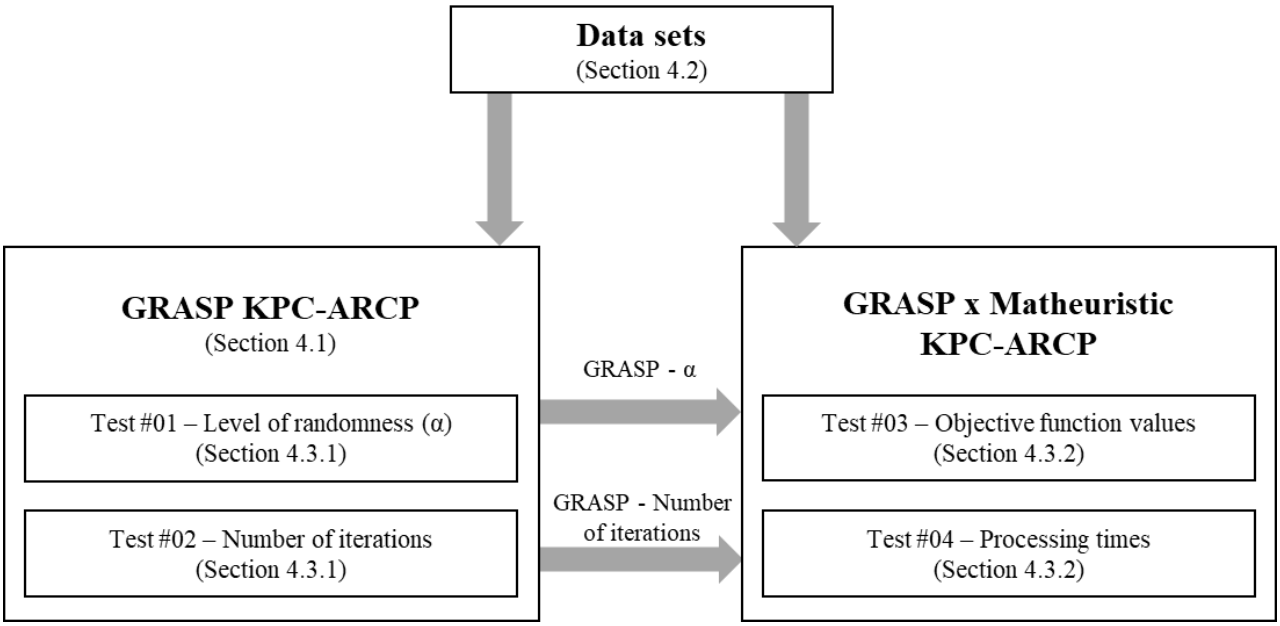


Figure 2 - Summary of the methodological approach applied in this paper

4.1 Description of proposed GRASP to KPC-ARCP

This paper proposes a metaheuristic to KPC-ARCP, the performance of which will be compared to the matheuristic approach by Akbari & Salman (2017a). The main advantages of a metaheuristic are that it does not require a commercial solver, and it is fast and capable of solving larger instances. There are three main reasons why we chose to use the GRASP metaheuristic, originally proposed by Feo & Resende (1995). As displayed in Table 1, this metaheuristic was used in other works in the context of road clearing, with various authors reporting a good performance. Additionally, GRASP is easy to calibrate because it has only two input parameters, namely the number of iterations and the level of randomness alpha (α). Finally, GRASP is easy to implement, and since each iteration is independent, it is easy to parallelize. This section aims to describe the structure of the GRASP that is utilized to compare with the matheuristic.

4.1.1 GRASP's structure

This section explains the overall structure of the GRASP to KPC-ARCP (see Figure 3). In real scenarios, the network is incomplete, i.e. there is no connection between every pair of vertices. Therefore, the preparation step of this algorithm is to find the shortest path between every pair of vertices using Dijkstra's algorithm (Dijkstra, 1959). Observe that every time that an edge is unblocked, the shortest paths may change because of the waiting times associated with the synchronization between the teams. In large-scale problems, it is impracticable to recalculate the paths at every iteration. Hence, we estimate the paths' duration as if every team would have to traverse and unblock every damaged road.

There are five inputs needed: number of teams, α (i.e. the level of randomness), number of iterations, the identification number of the depot, and the upper bound on the routes' duration (i.e. the time horizon). In summary, the algorithm creates a set of teams` routes in every iteration and compares the objective

function (OF) to the best one found so far. Note that since this is a maximization problem, the initial OF is set to zero, as shown in Figure 3. The algorithm stops after a pre-determined number of iterations.

The constructive heuristic, further detailed in Section 4.1.2, generates a feasible solution (i.e. the teams' routes). Based on the components reconnected by the teams, one can find the total prize collected (i.e., OF value). The prize collected in each solution is compared to a threshold value (best OF). If the total prize collected is greater or equal to the threshold value, the solution is set as the best one, and the value of the best OF is updated. Otherwise, the solution is discarded. This GRASP version has three outputs: best objective function value (best OF), the teams` routes and the list of edges that should be unblocked.

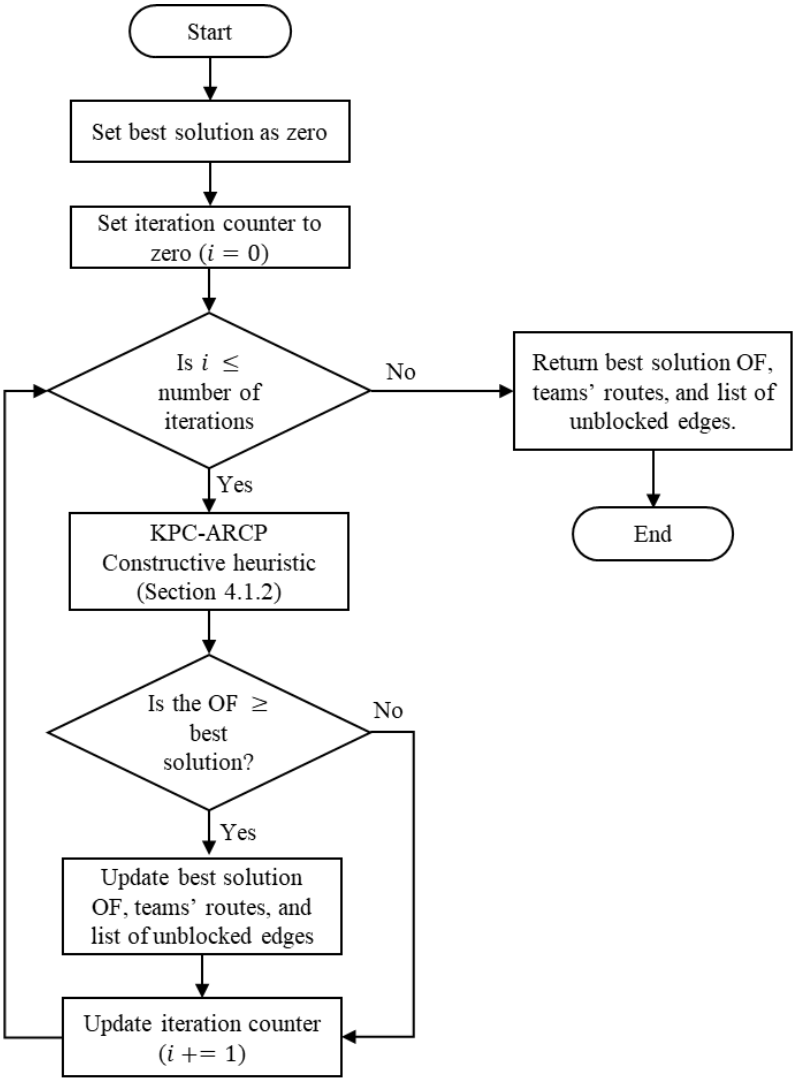


Figure 3 - Structure of GRASP for KPC-ARCP

4.1.2 KPC-ARCP constructive heuristic

The constructive heuristic creates a feasible solution to KPC-ARCP by building the team's route sequentially. The algorithm's goal is to reconnect the network while maximizing the total prize collected.

The heuristic constructs the routes by adding one component at a time and checking if the route is within the time limit. A route is feasible if its total duration is less than or equal to the time horizon.

The flow chart of the constructive heuristic is shown in Figure 4. The heuristic starts by identifying the separate components and the teams that are available at the depot. There are two stopping criteria. The heuristic ends when the network is reconnected or if all the teams have an assigned route.

The algorithm creates one route at a time if there is an unvisited component and at least one available team. The construction of a route is as follows. First, the depot is set as the start point. Then, the algorithm creates the Restricted Candidate List (RCL). This list has the α unvisited components with the highest prizes. Next, a candidate component from RCL is randomly selected following a uniform distribution.

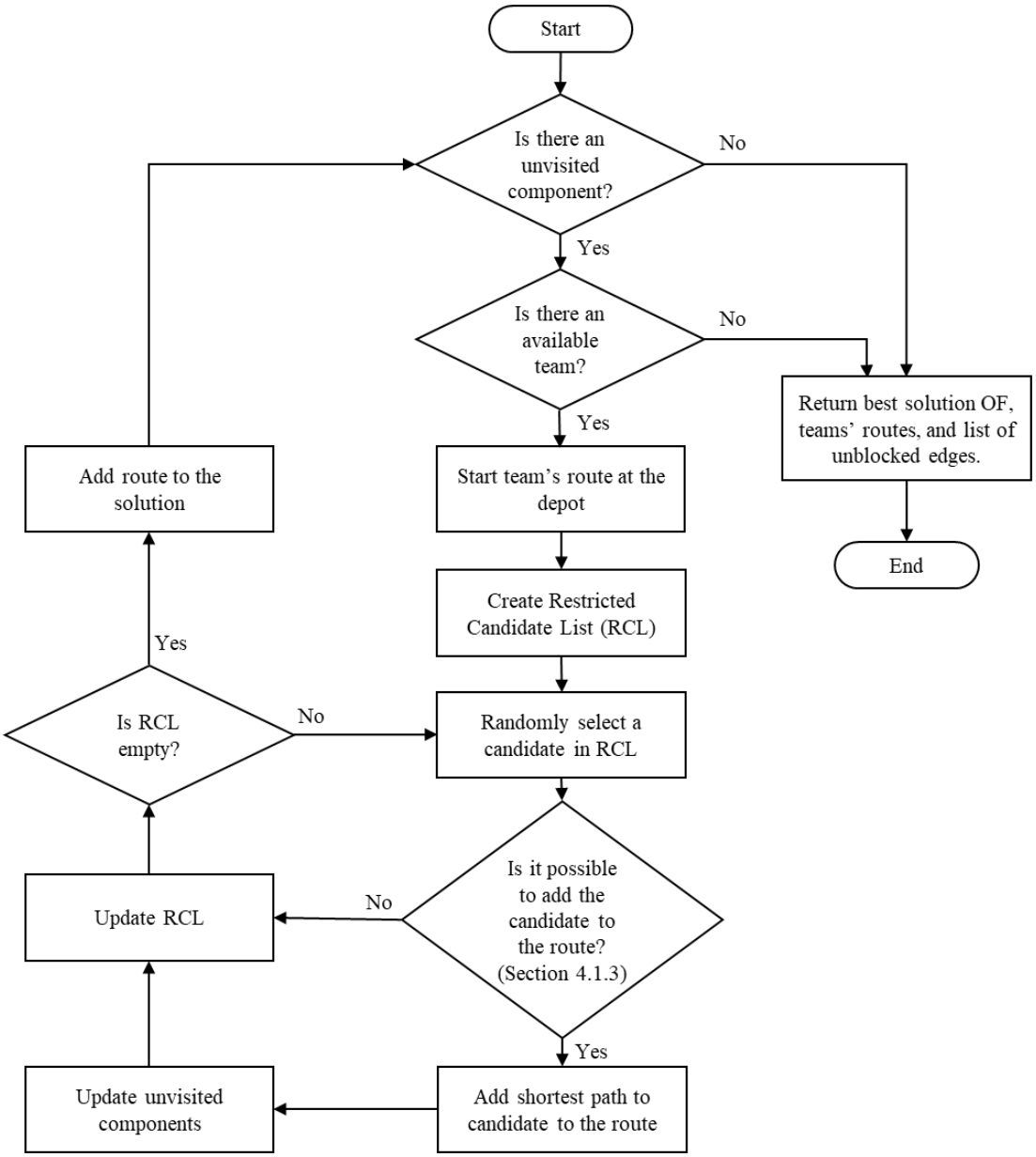


Figure 4 - Structure of the KPC-ARCP constructive heuristic

Since RCL is based on the prizes, it is not guaranteed that it is possible to visit each component within the time horizon. In addition, notice that the elements of RCL are components, whereas the route is a sequence of vertices. Therefore, it is necessary to use the shortest paths and durations calculated with Dijkstra's algorithm in the preliminary step, as mentioned in Section 4.1.1.

Before adding the path to the team's route, the algorithm checks its feasibility. The heuristic identifies all shortest paths between the last vertex in the team's route and all the vertices in the selected component. Then, it chooses the quickest path.

As an illustration, consider the network in Figure 5. The graph has four components (i.e. C_0, C_1, C_2 and C_3) as in Figure 5 (a), and the depot is in C_0. Figure 5 (b) represents the traversable roads (solid lines) and the blocked ones (dashed lines). Assume that a team is at the depot, and the selected component in RCL is C_2. The algorithm needs to find the shortest path between the depot and all the vertices in C_2 (i.e. 6,7,8 and 9). Some of the possible paths are [D,1,2,4,6], [D,3,7] and [D,3,10,8]. Assume that the path with the shortest duration is [D,3,10,8]. If the duration of this route is less than the time horizon, then this route is accepted. Otherwise, it fails the feasibility test, and the route is rejected.

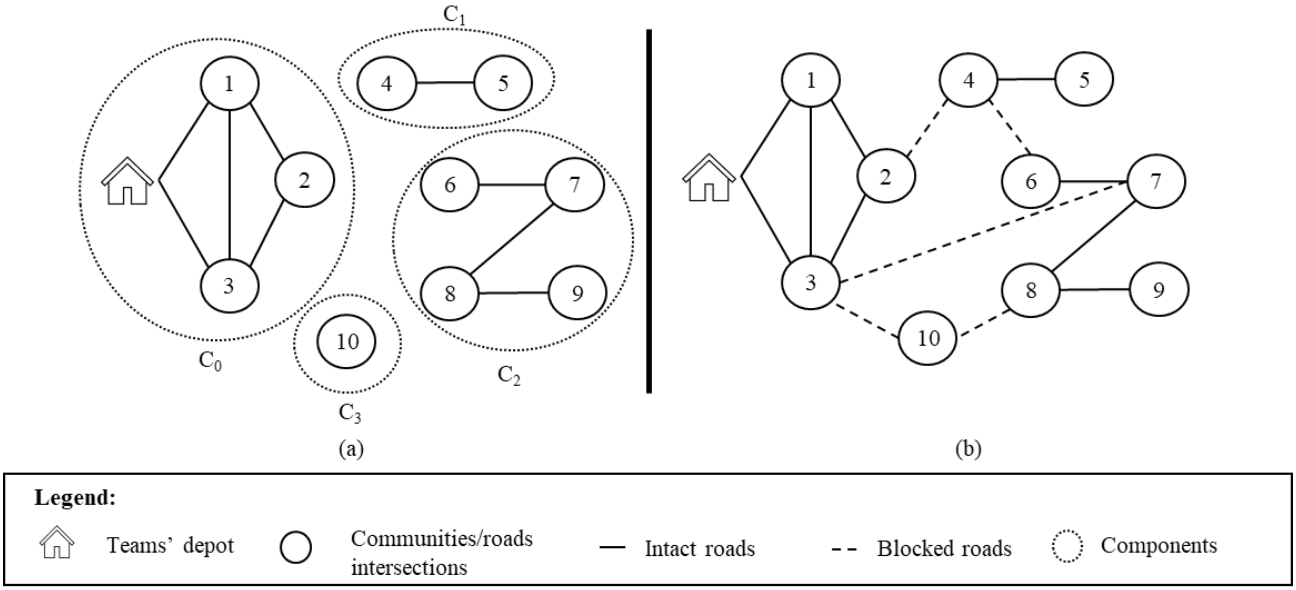


Figure 5 – Generic example of a road clearing scenario with (a) components representation and (b) road depiction

As mentioned, Dijkstra's algorithm is used to approximate the total duration of the path. Since this time estimate does not consider the synchronization between the teams, we use the algorithm described in Section 4.1.3 to calculate the actual duration of the route. If it is allowed to visit the selected component in the time horizon, the team's route is updated. In addition, it is verified whether an additional component is traversed. For example, in Figure 5 (b), to go from the depot to C_2, the shortest path crosses C_3 before C_2. Even though the candidate was C_2, the shortest path crosses two components.

In the case of a feasible route, all traversed components are set as visited. They are also removed from the RCL and replaced with another unvisited component. If it is impossible to visit the candidate, it is excluded from RCL without replacement, thus reducing RCL's size.

After some attempts to add a candidate to the route, the RCL becomes empty. This may be either because all candidates were reconnected or because it is not possible to add the candidates without exceeding the time horizon. Thus, the constructive heuristic for this team stops and the route is added to the final solution. Then, the algorithm checks if there are remaining unvisited components and available teams. If so, it proceeds to build a new route.

The constructive heuristic has three outputs. The algorithm identifies the teams' routes and the OF value. In addition, it establishes a list with each unblocked road, the team responsible for clearing it, and the time when it is unblocked.

4.1.3 Algorithm to calculate the duration of the team's route

This sub-section explains how to calculate the duration of each team's route. The input data is the route, the team, traversing and unblocking times, and the list of unblocked edges.

The synchronization between teams is guaranteed by the list of unblocked edges, which specifies when each team finishes clearing each edge being unblocked. For example, assume that edge (2,4) in Figure 5 (b) is scheduled to be unblocked by team A, and it will be cleared at hour 56. Then, while calculating the duration of team B, if the route of B crosses edge (2,4), the algorithm will make sure to wait until hour 56 to traverse the road.

The algorithm is shown in Figure 6, and it calculates the route's duration by analysing one edge at a time. The first step is to set the route's duration as zero. Then, the algorithm checks each arc and evaluates if it is blocked. If the edge is intact, the algorithm adds its traversing time to the duration and moves to the next edge. Otherwise, it checks if another team unblocked it.

If the current team is the first to cross the edge, the algorithm adds the unblocking and traversing times to the path duration and updates the list of unblocked edges. As the problem considers that a road is unblocked in both directions, the symmetric edge is also included in the list. However, if another team has already unblocked the edge, it is necessary to calculate the waiting time. In this case, both the waiting and the traversing times are added to the duration.

The waiting time is the difference between the completion time of the other team clearing the edge and the current duration of the team's route. The clearing completion time is available on the list of unblocked edges. As an example, assume that the duration of the team's B route is 48 hours. The next edge that team B is trying to traverse is (2,4), scheduled to be unblocked at hour 56 by team A. Then, the waiting time is the difference between 56 and 48 hours.

Finally, the output is the duration of the team's route and the updated version of the list of unblocked edges.

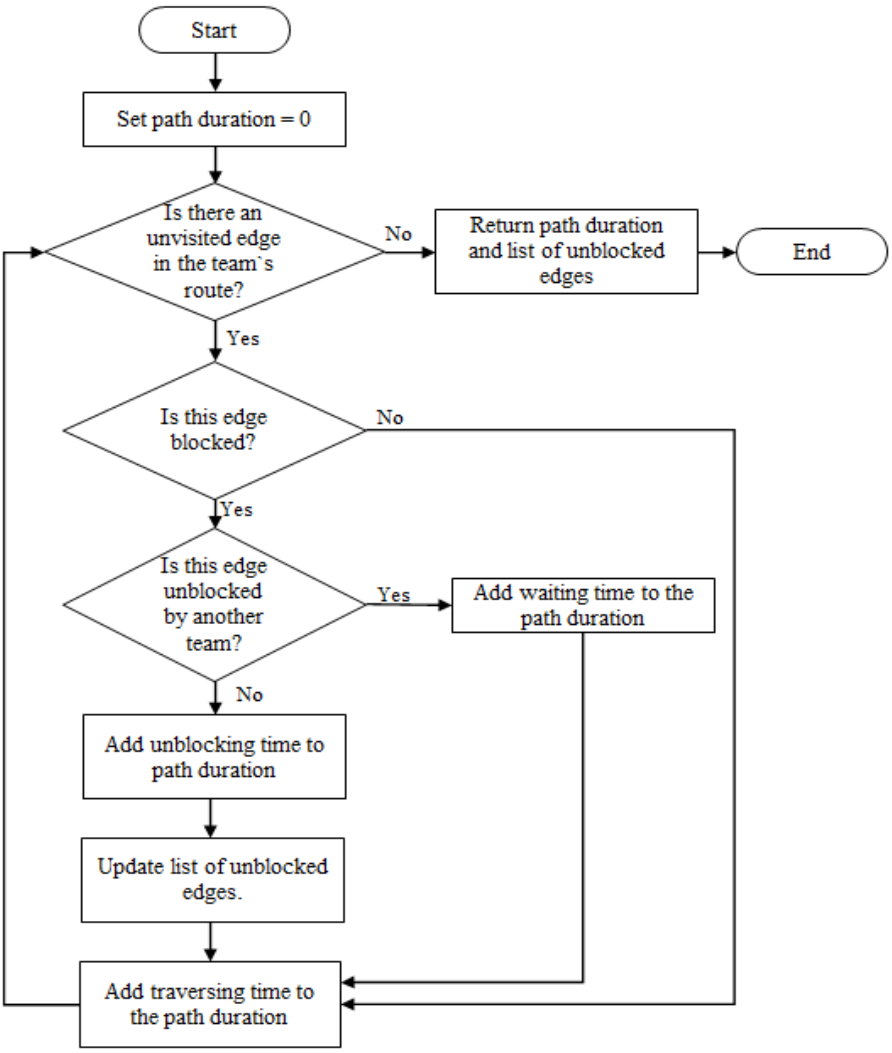


Figure 6 - Description of the algorithm to calculate the duration of each team's route

4.2 Instances generation

The comparison between GRASP and matheuristic to solve the KPC-ARCP is based on a set of randomly generated networks with increasing size. This sub-section covers the procedure to create these, with the required parameters and their descriptions given in Table 2. The intervals of the unblocking times and vertices' coordinates (i.e. plan dimension) are the same as in the work by Maya Duque & Sörensen (2011). The average degree of the graphs (i.e. the average number of vertex connections) follows Akbari & Salman (2017a). The data sets were generated in Python 3.8.5 and Networkx 2.3.

The initial graph represents the status of the roads before the disaster, and it is assumed that the road mesh is connected. In a real road network, not all vertices are connected to each other. Hence, it is necessary to develop an incomplete graph. The steps are as follows. First, the coordinates of n vertices are randomly generated with a uniform distribution, whose limits are the plan dimensions described in Table 2. Next, a vector with the degree of the n vertices is established. The degrees of the vertices are

randomly generated following the interval described in Table 2. Next, Networkx is used to create the graph's adjacency matrix which determines the vertices connections. If Networkx generates a disconnected graph, the components are linked with a randomly selected edge. Finally, the Euclidean distances are calculated.

Table 2 - Parameters used in the creation of data sets

Parameter	Description	Value
n	Instances size	50, 100, 200, 400, 800, 1000, 2000, 3000, 4000, 5000, 7000, 10000 vertices
(x, y)	Plan dimension	$(1000{,}1000)$
mc	Minimum number of components after the disaster	$[2, (0.3 * n)]$ components
b	The number of edges blocked in every iteration.	10% of the edge set
$[a, b]$	Uniform distribution interval (used to find the unblocking times).	$[0{,}100]$ hours
deg	Interval of the vertices' degree	[0,10]

The disaster breaks the initial network. Hence, it is necessary to determine the damaged roads and their unblocking times. First, the minimum number of components (mc) into which the graph will be divided is determined. Then, a batch of b edges is randomly blocked. An unblocking time is generated for each blocked edge, which follows a uniform distribution, specified in Table 2. After removing a set of edges, the algorithm counts the number of components. If there are more components than mc, the process will stop. Otherwise, the algorithm keeps blocking other batches of edges until there are more components than mc.

The description of the data sets, including the number of vertices, edges, damaged edges, and components, and the vertices` average degree, is given in Table 3. The average degree of the graphs is close to those of the cases tested in Akbari & Salman (2017a), which is approximately 5. As Akbari & Salman (2017a) demonstrated, the matheuristic performs very well on smaller instances, which are easy and fast to solve. Thus, three instances (A, B, and C) were created for 50, 100, and 200 sizes to support the analysis of the GRASP`s performance and the comparison with the matheuristic over smaller instances.

Table 3 – Description of the data sets used in the tests

Instance Name	Vertices	Edges	Damaged edges	Components	Average degree
50-A	50	206	76	7	3.92
50-B	50	264	52	4	4.98
50-C	50	252	50	3	5.06
100-A	100	522	358	26	5.00
100-B	100	568	114	7	4.9
100-C	100	484	192	10	4.75
200-A	200	1052	428	23	5.22
200-B	200	1022	612	41	4.81
200-C	200	1072	428	17	5.32
400-A	400	1940	884	45	4.95
800-A	800	4086	2754	214	5.06
1000-A	1000	4916	3150	265	4.74
2000-A	2000	10114	6232	473	4.92
3000-A	3000	15230	6650	364	4.97
4000-A	4000	20596	8240	389	5.03
5000-A	5000	25638	10256	544	5.01
7000-A	7000	35854	14340	730	4.95
10000-A	10000	51044	20416	1017	4.98

4.3 Test procedure

A total of four tests were employed to assess if GRASP performs better than matheuristic. The first two tests (described in Section 4.3.1) aim to calibrate the GRASP parameters (i.e. α and the number of iterations). The other parameters used in the tests are given in Table 4, and they are based on the information in Maya Duque & Sörensen (2011) and Akbari & Salman (2017a). In addition, it is assumed that the depot is the vertex with id zero. After defining the GRASP values, two tests are executed to evaluate the performance of the methods, as described in Section 4.3.2. The solution approaches were coded in Python 3.8.5, and the matheuristic was solved with Gurobi 9.1.2. The computational power used on all tests was an AMD Ryzen 9 3960X 16-core processor with 64GB of RAM.

4.3.1 Preliminary tests

Initially, two tests are executed to determine the GRASP parameters. The first test aims to define level of randomness (α), and the second test determines the number of iterations. The two tests happen simultaneously. All possible combinations of the three different percentages of α, number of iterations, and teams are tested with GRASP. The tests are executed three times for each of the instances due to the stochastic nature of the GRASP algorithm and to anticipate possible variations in the resulting performance. Then, the average objective function value and the average processing times are calculated. The parameters are selected based on a trade-off between the gap in the objective function value and the processing time. The gap in the objective function is calculated as follows: one hundred multiplied by

the difference between the GRASP and matheuristic objective function values, divided by the matheuristic objective function value.

4.3.2 Comparative tests

After identifying the value of α and the number of iterations, GRASP and the matheuristic are tested. Both solution methods are executed on networks up to 5,000 vertices and the OF value and processing times are recorded. Each approach is tested on cases with 4, 8 and 16 teams. In addition, GRASP is applied to sets with 7,000 and 10,000 vertices to test its feasibility on larger instances. The matheuristic is not tested on instances with 7,000 and 10,000 vertices given the trends shown in the results of smaller instances (Section 5.2).

Table 4 - Parameters used in the performance tests

Parameter	Description	Value
s	Speed of road clearing teams	50 km/h
k	Number of available teams in each depot	4,8,16
α	RCL size	25,50,75% of the number of components
i	Number of GRASP iterations	100,250,500,1000
P_q	Prize of each component	Total of vertices in the component

5 Results

In this paper, we compare the performance of GRASP and matheuristic on multiple data sets. The objective is to test the hypothesis that GRASP can solve more extensive networks in less time. The results related to the calibration of GRASP's parameters are presented in Section 5.1, whereas the comparison tests are in Section 5.2.

5.1 Preliminary tests

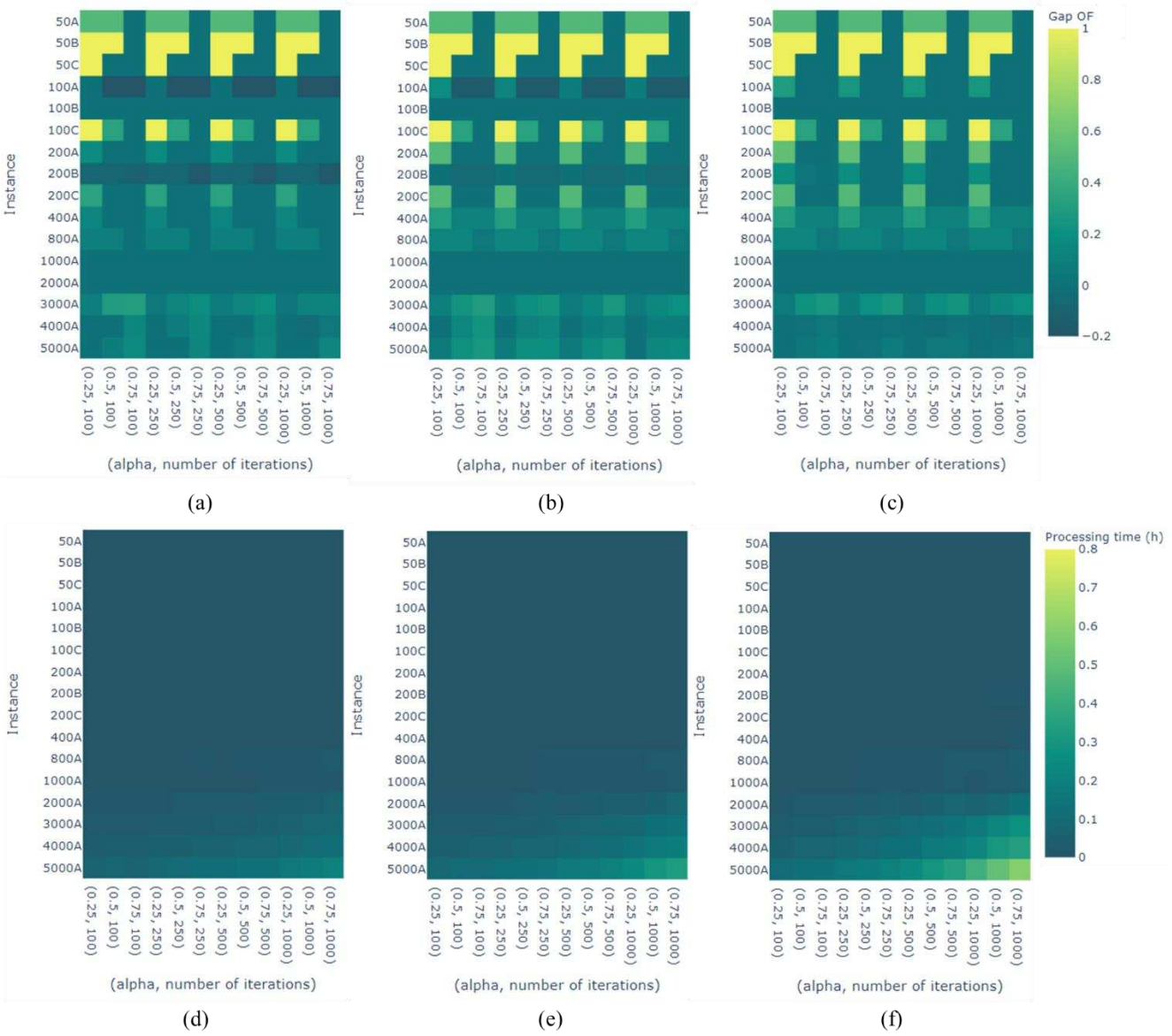
The purpose of the preliminary test is to calibrate GRASP's parameters considering a trade-off between the gap in the matheuristic solutions and the execution time. The results are available in Appendices B and C.

The results on the gap of the OF values are shown in Figure 7 (a-c). The horizontal axis is ordered according to the α values, and the vertical axis to the instance sizes. The cells' colour intensity relates to the gap in the OF value as depicted in the colour scale. Bright cells represent a high gap and darker cells a low one. As indicated in Figure 7 (a-c), the instances with less than 2,000 vertices should have an α value of 0.75. The other sets should have an α value of 0.25.

On the one hand, the number of iterations does not significantly impact the gap for instances with at most 2,000 vertices. Thus, we adopted 100 iterations as this results in a faster calculation time. On the other hand, the more extensive networks are affected by the selected number of iterations. Figure 7 (d-f) displays the processing times in hours of each one of the combinations of the parameters. As demonstrated in the figure, the processing time and the number of iterations increase proportionally.

Even though 1000 iterations are the slowest, the largest instance tested (i.e. 5000 vertices and 1000 iterations) runs in less than one hour. The number of iterations selected was 1000 because it gives the smallest objective function gap. Observe that 500 and 250 also have good performance and are suitable for cases that need a solution in half an hour.

In conclusion, the results of the preliminary tests indicate that for instances with less than 3,000 vertices, parameters choice of 0.75 and 100 iterations provide reasonable results. For other instances, parameters 0.25 and 1,000 iterations appear to be better.



(a) (b) (c)

(d) (e) (f)

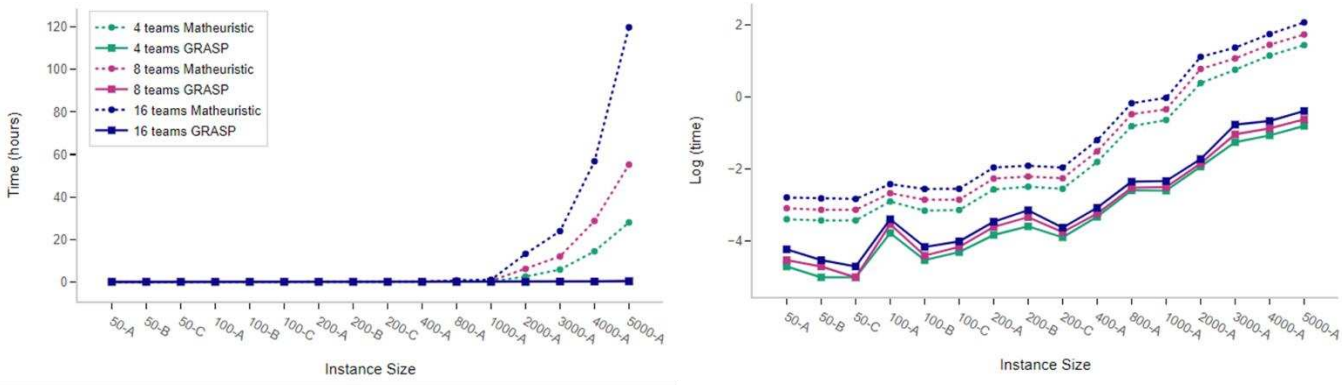
Figure 7 – Gap between GRASP and matheuristic in terms of objective function value (%) for (a) 4 teams, (b) 8 teams, and (c) 16 teams. Processing time in hours (d) 4 teams, (e) 8 teams, and (f) 16 teams

5.2 Comparative tests

The results of the comparative tests are shown in Figures 8 and 9. The gaps in the OF demonstrate that GRASP obtained better solutions than matheuristic in 8% of the cases, see Figure 8. The matheuristic found better results in 15% of the cases, and similar solutions in 77% of the cases. In the scenario with four teams, GRASP achieved the same or better results in all instances except 3000-A. In the best case, GRASP improved the OF value by 22%. In contrast, in the set of 400 vertices, GRASP's OF value is 10% lower than matheuristic. The average gap is approximately 0.22%, indicating that GRASP can obtain solutions which are practically as good as the matheuristic.

Figure 8 – Gap of the OF value (%) between GRASP and matheuristic.

The processing times are depicted in Figure 9. These results confirm our hypothesis that GRASP is faster. From the charts in Figure 9, it can be seen that matheuristic times (dashed lines) tend to increase considerably on large-scale data sets, leading to impractically long calculation times. On instances that GRASP (solid lines) solved in less than one hour, the matheuristic needed more than four days. Figure 9 (b) shows the times on a logarithmic scale, from which it is possible to visualize the difference in the execution times of GRASP.



(a) (b)

Figure 9 – Processing times of GRASP and matheuristic (a) in hours and (b) logarithmic scale

The final test verified that GRASP could process more extensive networks and the execution times are displayed in Table 5. GRASP solved the case with 16 teams in the network with 7,000 vertices in less than two hours and 10,000 vertices in less than three hours.

Table 5 – GRASP processing times on larger networks

Instance	4 teams	8 teams	16 teams
7,000	0.28 h	0.8 h	1.21 h
10,000	0.81 h	1.49 h	2.20 h

Overall, these results confirm the hypothesis that GRASP is faster and can solve larger networks than the matheuristic to KPC-ARCP, with a comparative accuracy in terms of the OF value.

6 Discussion

In this section, we discuss uncertainties associated with the choices made during the testing phase, describe some of the limitations of the metaheuristic approach, and suggest future work to expand the knowledge related to KPC-ARCP.

First, there is room for improvement in terms of methodology. Even though GRASP is very efficient, other common procedures applied in the literature as Ant Colony Optimization are applicable to the KPC-ARCP. In addition, in order to manage the stochastic nature of GRASP, we run each instance three times and used the average results in the comparison with the matheuristic. However, future studies can adopt other strategies such as choosing the lowest objective function value and the highest processing times of the three runs to guarantee that the worst performance of GRASP is as good as the matheuristic.

One limitation of this paper is that only one design choice on how to build the RCL in the constructive heuristic was tested. Thus, the design of RCLs to KPC-ARCP can be the topic of future work. For instance, in this paper, the RCL is created based on the objective function and thus on a single criterion: the components` prizes. A weighted combination of time, number of components implicitly visited, and distance could be used to determine the candidate components included in the RCL. This weighted combination could lead to improvements on GRASP's performance, but it requires a sensitivity analysis of the weights. Another improvement is to test the performance of the algorithm considering a different stop criterion for the construction of a team's route. In this paper, the components are removed from the RCL if they cannot be visited within the time horizon and the construction of the team's route stops when the RCL becomes empty. Other strategies could be implemented, such as stop building the team's route after failing to include a pre-determined number of components at the team's route.

Future research can focus on different heuristics for KPC-ARCP and improvements on GRASP. Instead of using a constructive heuristic and building each team's route separately, one could create a sequence of components to be visited and dividing it on the routes of a feasible team. In this case, only a certain number of routes would be kept as KPC-ARCP has a maximum number of available teams. Additionally, further studies can add local search operators to the constructive heuristics and check potential performance improvements. The GRASP metaheuristic can be improved by adopting other strategies for

the stop criterion and level of randomness parameter. In this paper, we adopted a constant value for alpha, but future studies can variate alpha along the iterations.

Another limitation is that we used randomly generated instances to compare GRASP and matheuristic. It is possible that the performance varies with other network structures. For example, the matheuristic's processing times may improve on graphs with a lower average degree (e.g. 3) or with a smaller set of damaged roads. Additionally, GRASP could be tested in dense graphs. It would be worthwhile to extend the tests to networks of realistic case studies and the set of Istanbul instances proposed by Akbari & Salman (2017a) to confirm that GRASP is more efficient in these instances as well.

As KPC-ARCP is a relatively new problem, there are many lines of research to further extend the problem formulation itself, and to test which solution algorithms provide optimal results in terms of solution accuracy and computing time. For example, the problem can be expanded to allow multiple start points, or to consider that the teams can be specialized. In addition, one can take into account the synchronization of the teams in a case in which aftershocks strike the network multiple times, so the damaged edges and unblocking times are modified over the time horizon. Another extension of the problem formulation can be the synchronization of road clearing teams' tasks with the distribution of relief supplies. Based on the results of the current work, it appears plausible that for future extensions of the problem formulation, GRASP can be selected as a solution method.

7 Conclusions

When natural disasters such as earthquakes occur, roads can be severely impacted, possibly resulting in the road mesh being disconnected. KPC-ARCP addresses such a problem, for which previous authors proposed a matheuristic solution method. Since the damaged network can be extensive, we tested the matheuristic method on large instances. Additionally, we suggested an alternative solution named GRASP to KPC-ARCP.

Two preliminary tests aimed to calibrate the parameters of GRASP. Then, the performance of both solution methods on instances with up to 5,000 vertices was evaluated. The tests compare the objective function and the processing times on cases with 4, 8 and 16 teams.

The study performed in this paper confirmed the hypothesis that GRASP is faster and can solve larger instances than matheuristic to KPC-ARCP. The results showed a low gap in objective function values between both methods, indicating that the results of GRASP are practically equally accurate. In addition, the tests demonstrated that GRASP was faster in all instances. The findings of this study have a number of important implications for future practice, while opening various avenues for future research. For example, researchers and practitioners can use GRASP to KPC-ARCP to solve road clearing models to give decision-makers valuable insights regarding their emergency plans.

The work in this study can be extended to the case with multiple start points or specialized teams, which are assigned to clear a specific subset of damaged edges. Another valuable extension is to test GRASP's

performance on real case networks. Finally, further research can focus on synchronizing road clearing activities to the distribution of relief supplies.

Acknowledgements

This work was supported by the Province of British Columbia and the Marine Environmental Observation, Prediction and Response [grant numbers 2-02-03-040.2, 2-02-03-041.3, 01-02-04-006]; and MITACS [grant number IT15323]. This financial support is gratefully acknowledged.

References

Ajam, M., Akbari, V., Salman, F.S., 2019. Minimizing latency in post-disaster road clearance operations. Eur. J. Oper. Res. 277, 1098–1112. https://doi.org/10.1016/j.ejor.2019.03.024

Akbari, V., Sadati, M.E.H., Kian, R., 2021a. A decomposition-based heuristic for a multicrew coordinated road restoration problem. Transp. Res. Part Transp. Environ. 95, 102854. https://doi.org/10.1016/j.trd.2021.102854

Akbari, V., Salman, F.S., 2017a. Multi-vehicle prize collecting arc routing for connectivity problem. Comput. Oper. Res. 82, 52–68. https://doi.org/10.1016/j.cor.2017.01.007

Akbari, V., Salman, F.S., 2017b. Multi-vehicle synchronized arc routing problem to restore post-disaster network connectivity. Eur. J. Oper. Res. 257, 625–640. https://doi.org/10.1016/j.ejor.2016.07.043

Akbari, V., Shiri, D., Sibel Salman, F., 2021b. An online optimization approach to post-disaster road restoration. Transp. Res. Part B Methodol. 150, 1–25. https://doi.org/10.1016/j.trb.2021.05.017

Asaly, A.N., Salman, F.S., 2014. Arc Selection and Routing for Restoration of Network Connectivity After a Disaster, in: Kara, B.Y., Sabuncuoglu, I., Bidanda, B. (Eds.), Global Logistics Management, Industrial Engineering Ser. Taylor & Francis Group, London, United Kingdom, p. 316.

Barbalho, T.J., Santos, A.C., Aloise, D.J., 2020. Metaheuristics for the work-troops scheduling problem. Int. Trans. Oper. Res. n/a. https://doi.org/10.1111/itor.12925

Berktaş, N., Kara, B.Y., Karaşan, O.E., 2016. Solution methodologies for debris removal in disaster response. EURO J. Comput. Optim. 4, 403–445. https://doi.org/10.1007/s13675-016-0063-1

Bruneau, M., Chang, S.E., Eguchi, R.T., Lee, G.C., O'Rourke, T.D., Reinhorn, A.M., Shinozuka, M., Tierney, K., Wallace, W.A., von Winterfeldt, D., 2003. A framework to quantitatively assess and enhance the seismic resilience of communities. Earthq. Spectra 19, 733–752. https://doi.org/10.1193/1.1623497

Çelik, M., 2016. Network restoration and recovery in humanitarian operations: Framework, literature review, and research directions. Surv. Oper. Res. Manag. Sci. 21, 47–61. https://doi.org/10.1016/j.sorms.2016.12.001

Dijkstra, E.W., 1959. A note on two problems in connexion with graphs. Numer. Math. 1, 269–271. https://doi.org/10.1007/BF01386390

Feo, T.A., Resende, M.G.C., 1995. Greedy Randomized Adaptive Search Procedures. J. Glob. Optim. 6, 109–133. https://doi.org/10.1007/BF01096763

Kasaei, M., Salman, F.S., 2016. Arc routing problems to restore connectivity of a road network. Transp. Res. Part E Logist. Transp. Rev. 95, 177–206. https://doi.org/10.1016/j.tre.2016.09.012

Kim, S., Shin, Y., Lee, G.M., Moon, I., 2018. Network repair crew scheduling for short-term disasters. Appl. Math. Model. 64, 510–523. https://doi.org/10.1016/j.apm.2018.07.047

Kovács, G., Tatham, P., 2007. An initial investigation into the application of the military sea-basing concept to the provision of immediate relief in a rapid onset disaster, in: Proceedings of the 18th Annual Conference of the Production and Operations Management Society. Presented at the POMS 18th Annual Conference, POMS - Production and Operations Management Society, Dallas, TX, United States.

Masson, A., 2014. The extratropical transition of Hurricane Igor and the impacts on Newfoundland. Nat. Hazards Dordr. 72, 617–632. https://doi.org/10.1007/s11069-013-1027-x

Maya Duque, P., Dolinskaya, I.S., Sörensen, K., 2016. Network repair crew scheduling and routing for emergency relief distribution problem. Eur. J. Oper. Res. 248, 272–285. https://doi.org/10.1016/j.ejor.2015.06.026

Maya Duque, P., Sörensen, K., 2011. A GRASP metaheuristic to improve accessibility after a disaster. Spectr. 33, 525–542. https://doi.org/10.1007/s00291-011-0247-2

Moreno, A., Alem, D., Gendreau, M., Munari, P., 2020a. The heterogeneous multicrew scheduling and routing problem in road restoration. Transp. Res. Part B Methodol. 141, 24–58. https://doi.org/10.1016/j.trb.2020.09.002

Moreno, A., Munari, P., Alem, D., 2020b. Decomposition-based algorithms for the crew scheduling and routing problem in road restoration. Comput. Oper. Res. 119, 104935. https://doi.org/10.1016/j.cor.2020.104935

Moreno, A., Munari, P., Alem, D., 2019. A branch-and-Benders-cut algorithm for the Crew Scheduling and Routing Problem in road restoration. Eur. J. Oper. Res. 275, 16–34. https://doi.org/10.1016/j.ejor.2018.11.004

National Police Agency of Japan, 2020. Police Countermeasures and Damage Situation associated with 2011Tohoku district - off the Pacific Ocean Earthquake. National Police Agency of Japan.

Norton, T.R., 2018. Lessons Learned in Disaster Debris Management of the 2011 Great East Japan Earthquake and Tsunami, in: Santiago-Fandiño, V., Sato, S., Maki, N., Iuchi, K. (Eds.), The 2011 Japan Earthquake and Tsunami: Reconstruction and Restoration: Insights and Assessment after 5 Years. Springer International Publishing, Cham, pp. 67–88. https://doi.org/10.1007/978-3-319-58691-5_5

Sahin, H., Kara, B.Y., Karasan, O.E., 2016. Debris removal during disaster response: A case for Turkey. Socioecon. Plann. Sci. 53, 49–59. https://doi.org/10.1016/j.seps.2015.10.003

Sakuraba, C.S., Santos, A.C., Prins, C., Bouillot, L., Durand, A., Allenbach, B., 2016. Road network emergency accessibility planning after a major earthquake. EURO J. Comput. Optim. 4, 381–402. https://doi.org/10.1007/s13675-016-0070-2

Sato, T., Suzuki, K., 2013. Impact of transportation network disruptions caused by the Great East Japan Earthquake on distribution of goods and regional economy. J. Jpn. Soc. Civ. Eng. 1, 507–515.

Tüzün Aksu, D., Özdamar, L., 2014. A mathematical model for post-disaster road restoration: Enabling accessibility and evacuation. Transp. Res. Part E Logist. Transp. Rev. 61, 56–67. https://doi.org/10.1016/j.tre.2013.10.009

Vodák, R., Bíl, M., Křivánková, Z., 2018. A modified ant colony optimization algorithm to increase the speed of the road network recovery process after disasters. Int. J. Disaster Risk Reduct. 31, 1092–1106. https://doi.org/10.1016/j.ijdrr.2018.04.004

Yan, S., Shih, Y.-L., 2012. An ant colony system-based hybrid algorithm for an emergency roadway repair time-space network flow problem. Transportmetrica 8, 361–386. https://doi.org/10.1080/18128602.2010.515550

Yan, S., Shih, Y.-L., 2007. A time-space network model for work team scheduling after a major disaster. J. Chin. Inst. Eng. 30, 63–75. https://doi.org/10.1080/02533839.2007.9671231

Appendix A – KPC-ARCP Sets and Parameters

Notation		Description
V	Set of vertices	The set of vertices represents the roads' intersections and the affected communities.
E	Set of edges	The set of edges represent the roads segments. This set is composed of intact, traversable, and blocked roads. The intact roads were not affected by the disaster, whereas the blocked ones were obstructed. After the team clears a blocked road, the road becomes traversable.
Q	Set of components	The set of components consists of groups of vertices that remained connected after the disaster takes place, and the initial graph becomes disconnected.
P	Component's prize	Each component has an associated prize which represents its priority weight.
K	Set of road clearing teams	Set of teams available to clear the blocked roads. The teams consist of the required machinery, heavy equipment, and workforce.
t	Time horizon	The time span of the problem starts right after the disaster occurs and ends at the time horizon.

Appendix B – GRASP - Gap Objective Function Value (%)

	Instance size / alpha	100 iterations			250 iterations			500 iterations			1000 iterations		
		25%	**50%**	**75%**	**25%**	**50%**	**75%**	**25%**	**50%**	**75%**	**25%**	**50%**	**75%**
4 teams	50-A	50%	50%	0%	50%	50%	0%	50%	50%	0%	50%	50%	0%
	50-B	100%	100%	0%	100%	100%	0%	100%	100%	0%	100%	100%	0%
	50-C	100%	0%	0%	100%	0%	0%	100%	0%	0%	100%	0%	0%
	100-A	0%	-22%	-22%	0%	-22%	-22%	0%	-22%	-22%	0%	-22%	-22%
	100-B	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
	100-C	100%	33%	0%	100%	33%	0%	100%	33%	0%	100%	33%	0%
	200-A	17%	0%	0%	17%	0%	0%	17%	0%	0%	17%	0%	0%
	200-B	-8%	-8%	-11%	-8%	-8%	-17%	-8%	-8%	-17%	-8%	-8%	-17%
	200-C	33%	0%	0%	33%	0%	0%	33%	0%	0%	33%	0%	0%
	400-A	13%	0%	0%	13%	0%	0%	13%	0%	0%	13%	0%	0%
	800-A	9%	9%	0%	9%	9%	0%	9%	9%	0%	9%	9%	0%
	1000-A	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
	2000-A	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
	3000-A	9%	29%	31%	4%	13%	18%	7%	13%	18%	2%	11%	11%
	4000-A	0%	0%	15%	0%	7%	15%	0%	0%	11%	0%	-4%	4%
	5000-A	0%	8%	17%	0%	0%	13%	0%	4%	13%	0%	0%	13%

	Instance size / alpha	100 iterations			250 iterations			500 iterations			1000 iterations		
		25%	**50%**	**75%**	**25%**	**50%**	**75%**	**25%**	**50%**	**75%**	**25%**	**50%**	**75%**
8 teams	50-A	50%	50%	0%	50%	50%	0%	50%	50%	0%	50%	50%	0%
	50-B	100%	100%	0%	100%	100%	0%	100%	100%	0%	100%	100%	0%
	50-C	100%	0%	0%	100%	0%	0%	100%	0%	0%	100%	0%	0%
	100-A	18%	-15%	-15%	13%	-15%	-15%	8%	-15%	-15%	8%	-15%	-15%
	100-B	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
	100-C	100%	33%	0%	100%	33%	0%	100%	33%	0%	100%	33%	0%
	200-A	50%	0%	0%	50%	0%	0%	50%	0%	0%	50%	0%	0%
	200-B	0%	-6%	-6%	0%	-6%	-6%	0%	-6%	-6%	0%	-6%	-6%
	200-C	50%	0%	0%	50%	0%	0%	50%	0%	0%	50%	0%	0%
	400-A	30%	10%	10%	30%	10%	10%	30%	10%	10%	30%	10%	10%
	800-A	13%	13%	4%	13%	13%	4%	13%	13%	4%	13%	13%	4%
	1000-A	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
	2000-A	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
	3000-A	9%	23%	29%	8%	19%	23%	7%	13%	25%	7%	16%	20%
	4000-A	0%	15%	23%	0%	10%	15%	0%	8%	10%	0%	8%	8%
	5000-A	6%	21%	25%	6%	19%	19%	6%	17%	23%	0%	15%	17%

	Instance size / alpha	100 iterations			250 iterations			500 iterations			1000 iterations		
		25%	50%	75%	25%	50%	75%	25%	50%	75%	25%	50%	75%
16 teams	50-A	50%	50%	0%	50%	50%	0%	50%	50%	0%	50%	50%	0%
	50-B	100%	100%	0%	100%	100%	0%	100%	100%	0%	100%	100%	0%
	50-C	100%	0%	0%	100%	0%	0%	100%	0%	0%	100%	0%	0%
	100-A	29%	0%	0%	25%	0%	0%	25%	0%	0%	25%	0%	0%
	100-B	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
	100-C	100%	33%	0%	100%	33%	0%	100%	33%	0%	100%	33%	0%
	200-A	55%	0%	0%	55%	0%	0%	55%	0%	0%	55%	0%	0%
	200-B	19%	2%	0%	19%	0%	0%	19%	0%	0%	19%	0%	0%
	200-C	50%	0%	0%	50%	0%	0%	50%	0%	0%	50%	0%	0%
	400-A	30%	10%	10%	30%	10%	10%	30%	10%	10%	30%	10%	10%
	800-A	13%	13%	4%	13%	13%	4%	13%	13%	4%	13%	13%	4%
	1000-A	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
	2000-A	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%	0%
	3000-A	7%	22%	26%	8%	20%	25%	6%	19%	23%	7%	15%	20%
	4000-A	0%	1%	4%	0%	0%	1%	0%	-3%	1%	0%	-4%	0%
	5000-A	0%	3%	8%	0%	1%	7%	0%	4%	4%	0%	1%	4%

Appendix C – GRASP – Processing times (seconds)

	Instance size / alpha	100 iterations			250 iterations			500 iterations			1000 iterations		
		25%	50%	75%	25%	50%	75%	25%	50%	75%	25%	50%	75%
4 teams	50-A	0	0	0	0	0	0	0	0	0	0	1	1
	50-B	0	0	0	0	0	0	0	0	0	0	0	0
	50-C	0	0	0	0	0	0	0	0	0	0	0	0
	100-A	0	1	1	1	1	2	2	2	3	4	5	6
	100-B	0	0	0	0	0	0	0	0	0	0	1	1
	100-C	0	0	0	0	0	0	0	1	1	0	1	2
	200-A	0	0	1	1	1	1	1	2	2	2	3	4
	200-B	1	1	1	1	2	2	3	3	4	5	7	8
	200-C	0	0	0	1	1	1	1	1	2	2	3	3
	400-A	1	2	2	2	3	3	3	4	5	5	8	10
	800-A	6	8	9	10	13	18	17	23	33	30	43	62
	1000-A	7	8	9	10	12	14	13	18	23	21	32	41
	2000-A	31	38	43	42	55	74	58	86	118	92	143	211
	3000-A	74	82	83	101	106	120	129	155	170	204	239	280
	4000-A	108	147	155	166	188	202	222	247	267	315	365	400
	5000-A	231	286	217	318	343	381	412	462	477	578	642	711

		100 iterations			250 iterations			500 iterations			1000 iterations		
	Instance size / alpha	**25%**	**50%**	**75%**	**25%**	**50%**	**75%**	**25%**	**50%**	**75%**	**25%**	**50%**	**75%**
8 teams	50-A	14	0	0	0	0	0	0	0	1	1	1	1
	50-B	0	0	0	0	0	0	0	0	0	0	0	0
	50-C	0	0	0	0	0	0	0	0	0	0	0	0
	100-A	0	1	1	1	2	3	2	5	5	4	9	10
	100-B	0	0	0	0	0	0	0	1	1	1	1	1
	100-C	0	0	0	0	0	1	0	1	1	1	2	2
	200-A	0	1	1	1	2	2	1	3	4	3	6	7
	200-B	1	1	2	2	3	4	5	6	8	9	13	15
	200-C	0	1	1	1	1	1	1	2	3	2	5	5
	400-A	1	2	2	2	3	4	4	6	8	7	11	15
	800-A	7	8	11	12	16	23	21	30	43	38	56	83
	1000-A	8	10	11	12	16	21	18	27	36	31	48	67
	2000-A	34	43	52	51	68	93	77	111	158	131	195	292
	3000-A	91	105	112	136	154	176	209	238	286	337	416	505
	4000-A	170	181	202	221	248	267	311	364	406	490	595	679
	5000-A	314	344	361	415	468	491	595	657	725	878	1039	1185

1

		100 iterations			250 iterations			500 iterations			1000 iterations		
	Instance size / alpha	**25%**	**50%**	**75%**	**25%**	**50%**	**75%**	**25%**	**50%**	**75%**	**25%**	**50%**	**75%**
16 teams	50-A	16	0	0	0	0	0	1	1	1	1	2	2
	50-B	0	0	0	0	0	0	0	0	0	1	1	1
	50-C	0	0	0	0	0	0	0	0	0	1	1	1
	100-A	1	1	1	2	3	4	3	7	7	6	13	14
	100-B	0	0	0	0	0	1	1	1	1	1	2	2
	100-C	0	0	0	0	1	1	1	1	2	1	3	3
	200-A	1	1	1	1	3	3	2	5	6	4	10	11
	200-B	1	2	3	3	5	6	6	10	12	12	19	24
	200-C	1	1	1	1	2	2	2	3	4	4	6	7
	400-A	2	2	3	3	5	6	5	9	12	10	17	24
	800-A	9	12	16	17	24	36	30	45	68	57	87	132
	1000-A	10	13	17	17	25	34	29	44	63	52	83	120
	2000-A	42	54	69	71	97	134	117	165	241	210	303	458
	3000-A	121	146	160	210	247	291	349	423	515	627	775	965
	4000-A	196	237	257	309	374	416	474	603	684	791	1065	1218
	5000-A	367	419	456	561	654	737	869	1057	1209	1497	1830	2114

2