A GRASP heuristic for the Cooperative Communication Problem in Ad Hoc Networks

Clayton Commander* Carlos A.S. Oliveira † Panos M. Pardalos*
Mauricio G.C. Resende‡

*Department of Industrial and Systems Engineering, University of Florida
303 Weil Hall, Gainesville, FL 32611.
{clayton8,pardalos}@ufl.edu

†School of Industrial Engineering and Management, Oklahoma State University
322 Engineering North, Stillwater, OK 74078.
coliv@okstate.edu

‡Internet and Network Systems Research Center, AT&T Labs Research
Room C241, 180 Park Avenue, Florham Park, NJ 07932 USA.
mgcr@research.att.com

1 Introduction

Ad hoc networks are composed of a set of wireless units that can communicate directly, without the use of a pre-established server infrastructure. In this respect, ad hoc systems are fundamentally different from traditional cellular systems, where each user has an assigned base-station which connects it to the wired telephony system. In an ad hoc network, each client has the capacity of accessing network nodes that are within its reach. This connectivity model allows the existence of networks without a predefined topology and which change every time a node changes position.

Due to this inherent variability, ad hoc networks present serious challenges for the design of efficient protocols: the maximization of parameters in such networks is subject to the lack of global information and optimal solutions may be short-lived, due to the dynamics of user positions and connectivity status.

We study a problem of coordinating wireless users involved in a task that requires going from an initial location to a target location. The problem consists in maximizing the amount of connectivity among a set of users, subject to constraints on the maximum distance traveled by users, as well as restrictions on what types of movement can be performed. The resulting problem, called the *cooperative communication problem in mobile ad hoc networks* (CCPM) problem, is formally defined next.

1.1 Problem Definition

An ad hoc network is composed of a set of autonomous clients that can connect to each other using their own wireless capabilities. This includes using scarce resources such as computational processing and battery power. We model this using a special type of graph called a *unit graph*. A unit graph is a planar graph $G = (V,E)$, with associated positions for each node $v \in V$. If we let $d : V \times V \rightarrow \mathbb{R}$ be the Euclidean distance function, then we can state the main property of unit graphs: an edge occurs between nodes v and w of G if only if $d(v,w) \leq 1$. Unit graphs occur as a natural model in ad hoc networks and are used in this paper to represent the set of configurations of nodes that share connections.

Let $G = (V,E)$ be a graph representing the set of valid positions for network clients. This graph has the property that each node is connected only to nodes that can be reached in one unit of time. Therefore, the graph can be used to represent all possible trajectories of a node, and each such trajectory is a path $\mathcal{P} = \{v_1, \ldots, v_k\}$, where $v_1 \in V(G)$ is the starting node, and $v_k \in V(G)$ is the destination node. We consider also a set U of wireless units, a set of initial positions S, with $|S| = |U|$ and $S \subseteq V(G)$, and a set of destinations D, with $|D| = |U|$ and $D \subseteq V(G)$. We assume that to perform its task, each wireless unit $u_i \in U$ starts from a position $s_i \in S$, and moves to position $d_i \in D$. We are given a limit T such that all units must reach their destinations by time T.

The trajectory of users in the system occurs as follows. Let $N(v) \subseteq V(G)$ be the set of nodes in the neighborhoods of v, i.e., the set of nodes $w \in V(G)$ such that $(v,w) \in E(G)$. Let $p_t : U \rightarrow V(G)$ be a function returning the position of a wireless unit at time t. Then, at each time step t, a wireless unit $u \in U$ can stay in its previous position $p_{t-1}(u)$ or move to one of its neighbors $v \in N(u)$. That is, at time step t, position $p_t(u) \in p_{t-1}(u) \cup N(u)$.

Let $\{\mathcal{P}_i\}_{i=1}^{k}$,where $k = |U|$, be the set of paths representing the trajectories of the wireless units in U (obviously, the first node of $\mathcal{P}_i$ is s_i, and its last node is d_i). Let L_i, for $i \in \{1, \ldots, |U|\}$, be a threshold on the total costs of path $\mathcal{P}_i$. Thus, we require that for each $\mathcal{P}_i = \{v_1, \ldots, v_{n_i}\}$, the constraints

$$\sum_{j=2}^{n_i} d(v_{j-1}, v_j) \leq L_i \qquad \text{for each } \mathcal{P}_i = \{v_1, \ldots, v_{n_i}\} \tag{1.1}$$

be satisfied.

We define the *cooperative communication problem in mobile ad hoc networks* (CCPM) as follows. Given a graph $G = (V,E)$, a set U of users, a set $S \subseteq V(G)$ of starting nodes, a set $D \subseteq V(G)$ of destination nodes, a maximum time T, and distance thresholds L_i, for $i \in \{1, \ldots, |U|\}$, then a feasible solution for the CCPM consists of a set of positions $p_t(u)$, for $t \in \{1, \ldots, T\}$, and $u \in U$, such that the initial position $p_1(u) = s(u)$ for $u \in U$, the final position $p_T(u) = d(u)$ for $u \in U$, the moves are given by $p_t(u) \in p_{t-1}(u) \cup N(u)$, and the inequalities (1.1) are satisfied. The objective is to maximize the connectivity of users in U over the time horizon, i.e.:

$$\max \sum_{t=1}^{T} \sum_{u,v \in U} c(p_t(u), p_t(v)),$$

where $c : V \times V \rightarrow \{0,1\}$ is a function returning 1 iff $d(p(u), p(v)) \leq 1$. This is an NP-hard problem, as proved in [5] (see this paper for additional information on the problem).

```
1   c* ← ∞
2   while stopping criterion not satisfied do
3     s ← ConstructSolution()
4     s ← LocalSearch(s)
5     if cost(s) > c* then
6       s* ← s
7       c* ← cost(s)
8     end
9   end
10  return s*
```

Figure 2.1: GRASP for maximization

2 GRASP for the CCPM

GRASP (greedy randomized adaptive search procedure) [3] is a metaheuristic that has been used with success to provide solutions for several difficult combinatorial optimization problems [4]. The objective of GRASP is to efficiently probe different parts of the set of feasible solutions of a combinatorial optimization problem. Solutions are selected from the search space based on the quality of the objective function. The selected solutions are subsequently optimized using a local search algorithm, resulting usually in a solution with good quality. A pseudo-code for GRASP is presented in Figure 1.

Construction Phase. The first task in a GRASP algorithm is to create solutions that have good fitness according to the objective function considered. To do this, GRASP uses a iterative method that selects candidates for the solution according to a greedy criterion. A greedy function g can be used to determine a set of candidate elements that can most improve the objective function, considering only the local effect on the solution. The constructor in GRASP uses this information to construct a *restricted candidate list* (RCL) with elements that would improve the current partial solution. The actual selection is performed based on a parameter α, which is usually determined empirically or randomly, and the RCL list is created with a fraction α of the available elements. The element selected in each iteration is taken randomly from the RCL. This is done in order to improve the diversity of the created solution, without sacrificing much on the quality of the final solution. The described steps are summarized in the pseudo-code in Figure 2.3.

To construct solutions for the CCPM problem, we use a strategy based on decomposing the total scheduling of trajectories into several steps, according to the number of clients of the wireless ad hoc network. At each iteration of the constructor we schedule the trajectory of a new client, and therefore we have $|U|$ major iterations.

The method used during each iteration of the constructor consists in using shortest paths to link the source-destination pairs. Therefore, the initialization step consists in computing all shortest paths between all pairs of nodes (s_i, d_i), for $i \in \{1, \ldots, |U|\}$. Notice that this can be done in $O(|V(G)|^3)$ time with Floyd's algorithm, for example. Then, we set up the initial partial solution with the shortest path $\mathcal{P}_i$, where $i \in \{1, \ldots, |U|\}$ is selected with uniform probability. In the while loop of lines 6–16, which is executed whenever there is a source-destination pair that has not yet been scheduled, we consider the selection of a new path to be added to the solution. The first step is to create a list of paths ordered

```
1   for all pairs (s_i, d_i), s_i ∈ S, d_i ∈ D do
2     P_i ← shortest-path(s_i, d_i)
3   end
4   Schedule a user u_i ∈ U using shortest path P_i
5   S ← {i}
6   while there is a user u ∈ U, with u ∉ S do
7     L ← ∅
8     for all pairs (s_i, d_i) ∈ S × D such that i ∉ S do
9       Add P_i into L in decreasing order of number of
          connections resulting from its addition
10    end
11    Get random α ∈ [0, 1]
12    RCL ← top α fraction of L
13    Select a path P_i from RCL
14    Add P_i to the solution
15    S ← S ∪ {i}
16  end
```

Figure 2.2: Greedy randomized constructor for CCPM

according to a greedy function. The function $g : \{\mathcal{P}_i\}_{i\in\{1,\ldots,u\}} \rightarrow \mathbb{Z}_+$ used by the construction returns the number of connections resulting from the addition of a path $\mathcal{P}_i$ to the current set of paths.

The list L is then used to create the list of restricted candidates in following way. Select a value for the parameter α in the interval $[0,1]$ (notice that while α can be determined using several methods, we have decided to use a simple implementation). The restricted candidate list is formed by the α fraction of best elements stored in L. Finally, a candidate path $\mathcal{P}_i$ is selected from the RCL using a uniform distribution, and added to the solution — this is represented in the pseudo-code by setting S to $S \cup \{i\}$.

Improvement Phase The next phase of GRASP has the objective of improving the solution created by the greedy randomized constructor. There are several options for implementation of this local search phase, including gradient descent methods, 2-opt swap based methods, and even the use of other metaheuristics. In the implementation for the CCPM, we decided to use a steepest decent method, where the objective is to improve the solution as much as possible, until a local solution is found. This is described in the following pseudo-code.

In a local search algorithm, a neighborhood is defined for each solution. In the case of the CCPM, given a solution s, the neighborhood $N(s)$ of s consists of all feasible solutions that differ from s in exactly one trajectory. Clearly, the number of positions where a new path could be inserted in a solution is equal to $|U|$. However, the number of possible paths between two points can become very large, and therefore this is a neighborhood with exponential size. To avoid searching all the elements of the neighborhood of a solution, we check only $|U|$ possible neighbors at each iteration of the local search algorithm.

In the pseudo-code in Figure 2.3, starting from an initial solution passed as argument, the neighborhood of each current solution is explored. This is done until a local optimal solution is found. However,

```
1   compute cost c of current solution
2   while solution is not locally optimal and niter < Miter do
3     for all pairs (s_i, d_i), with i ∈ {1, ..., |U|} do
4       Remove current path P_i from solution
5       Use randomized DFS algorithm to find path P from s_i to d_i
          such that d(P) ≤ D_i
6       compute cost c' of new solution
7       if c' is better than c then
8         c ← c'
9       else
10        Revert to previous path P_i
11      end
12    end
13    niter ← niter + 1
14  end
```

Figure 2.3: Local Search for CCPM

as the neighborhood has exponential size, we limit the number of iterations to the value *Miter*, which provides an empirical upper bound on the running time. The mechanism of local perturbation is implemented in the following way. At each iteration, one of the source-destination pairs (s_i, d_i) is selected, and a new trajectory is created linking s_i to d_i. The path is computed using a randomized version of the *depth first search* (DFS) procedure. The difference of this to the original version of DFS is that the child node selected at each iteration is chosen according to an uniform distribution. This procedure takes time $O(m)$, and therefore it allows us to compute efficiently a new substitute path from s_i to d_i.

3 Experimental Results

In our study, the proposed GRASP was tested on 60 unit graphs ranging from 50 to 100 nodes with radii of communication ranging from 20 to 50 miles. Results are given with the number of mobile agents ranging from 10 to 50. The test cases were created by a generator from previous work by Butenko et al. [1, 2]. on the BROADCAST SCHEDULING problem. Computational experiments were performed on a PC with a 2.8 GHz processor and 1 GB of main memory. Our code was written in FORTRAN; Schage's random number generator was used with a seed of 270001 for all cases [6]. The GRASP performed 100 iterations before stopping.

The numerical results are summarized in Table 1. The solutions for each instance are the averages of 5 unit graphs with the same number of nodes and radius of communication. As one would expect, for each instance the average solution increases as the communication range increases. Likewise, we see that more agents contribute to higher objective function values. The computation time also tends to increases with the size of the graph and number of agents being routed. However, for most instances the average solution time was 1.45s with the exception of the 100 node graphs configuring trajectories for 50 agents. For these cases, the average solution was found in 18.57s. The heuristic proved to be very robust in that it was able to efficiently solve a wide variety of test instances. GRASP provided high quality solutions in a fraction of the time required by the pure IP solver, while configuring trajectories

Table 1: Results of GRASP for CCPM.

Instance	Nodes	Radius	Agents	Avg. Soln	Avg. Time	Agents	Avg. Soln	Avg. Time	Agents	Avg. Soln	Avg. Time
1	50	20	10	63.6	.052	15	152	.238	25	414.66	1.0
2	50	30	10	83.8	.127	15	182.2	.228	25	516.2	1.33
3	50	40	10	95.4	.238	15	228.6	.303	25	695	1.04
4	50	50	10	115.4	.314	15	275.8	.616	25	797.4	1.90
5	75	20	10	76.8	.115	20	270.2	1.02	30	575.2	3.04
6	75	30	10	85.8	.362	20	299.6	1.328	30	725.4	3.93
7	75	40	10	96.4	.452	20	386	1.54	30	862.6	3.45
8	75	50	10	125	.907	20	403.2	1.26	30	1082.4	2.19
9	100	20	15	113.6	.899	25	333.4	3.322	50	1523.2	15.91
10	100	30	15	166.2	.938	25	511.2	2.93	50	1901.4	17.62
11	100	40	15	203.4	2.52	25	600.6	3.31	50	2539.2	16.58
12	100	50	15	255.8	1.24	25	756.8	4.44	50	3271.2	24.15

for up to 10 times as many agents.

References

[1] S. Butenko, C. Commander, and P. Pardalos. On the performance of heuristics for broadcast scheduling. In *Theory and Algorithms for Cooperative Systems*. World Scientific, 2004.

[2] C. W. Commander, S. I. Butenko, P. M. Pardalos, and C. A. Oliveira. Reactive GRASP with path relinking for the broadcast scheduling problem. In *Proceedings of the 40th Annual International Telemetry Conference*, pages 792–800, 2004.

[3] T. Feo and M. Resende. Greedy randomized adaptive search procedures. *Journal of Global Optimization*, 6:109–133, 1995.

[4] P. Festa and M. Resende. GRASP: An annotated bibliography. In C. Ribeiro and P. Hansen, editors, *Essays and surveys in metaheuristics*, pages 325–367. Kluwer Academic Publishers, 2002.

[5] C. A. Oliveira and P. M. Pardalos. Ad hoc networks: Optimization problems and solution methods. In D.-Z. Du, M. Cheng, and Y. Li, editors, *Combinatorial Optimization in Communication Networks*. Kluwer, Dordrecht, 2005. In press.

[6] L. Schrage. A more portable FORTRAN random number generator. *ACM Transactions on Mathematical Software*, 5:132–138, 1979.