European Journal of Operational Research xxx (xxxx) xxx

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

Solving the maximum edge disjoint path problem using a modified Lagrangian particle swarm optimisation hybrid

Jake Weiner[a,*], Andreas T. Ernst[b], Xiaodong Li[a], Yuan Sun[a], Kalyanmoy Deb[c]

[a] School of Science, RMIT University, 124 La Trobe Street, Melbourne, Australia
[b] School of Mathematics, Clayton Campus, Monash University, Melbourne, Australia
[c] Department of Electrical and Computer Engineering, Michigan State University, Michigan, East Lansing, MI 48864, USA

ARTICLE INFO

Article history:
Received 11 November 2019
Accepted 4 January 2021
Available online xxx

Keywords:
Combinatorial optimization
Large scale optimization
Metaheuristics

ABSTRACT

This paper presents a new method to solve the Maximum Edge Disjoint Paths (MEDP) problem. Given a set of node pairs within a network, the MEDP problem is the task of finding the largest number of pairs that can be connected by paths, using each edge within the network at most once. We present a heuristic algorithm that builds a hybridisation of Lagrangian Relaxation and Particle Swarm Optimisation, referred to as LaPSO. This hybridisation is combined with a new repair heuristic, called Largest Violation Perturbation (LVP). We show that our LaPSO method produces better heuristic solutions than both current state-of-the-art heuristic methods, as well as the primal solution found by a standard Mixed Integer Programming (MIP) solver within a limited runtime. Significantly, when run with a limited runtime, our LaPSO method also produces strong bounds which are superior to a standard MIP solver for the larger instances tested, whilst being competitive for the remainder. This allows our LaPSO method to prove optimality for many instances and provide optimality gaps for the remainder, making it a "quasi-exact" method. In this way our LaPSO algorithm, which draws on ideas from both mathematical programming and evolutionary algorithms, is able to outperform both MIP and metaheuristic solvers that only use ideas from one of these areas.

1. Introduction

In communication networks there often exist multiple information requests that need to be facilitated simultaneously across the network. In many applications it is important to ensure that no two connection paths within the network interfere with one another, resulting in multiple disjoint paths. The task of connecting the largest number of requesting and transmitting node pairs, often termed commodities, without any two paths sharing any edges defines the Maximum Edge Disjoint Path (MEDP) problem. The MEDP problem can be found in a wide array of applications. In Optical Networks, different paths using the same wavelength are not allowed to share physical links (Raghavan & Upfal, 1994). In Very Large Scale Integration (VLSI) design, it is a requirement that wire paths do not interfere with each other (Chan, Chin, & Ting, 2003; Gerez, 1998). Edge Disjoint Paths also play an important role in Ad Hoc Networks, to reduce energy consumption (Sumpter, Burson, Tang, & Chen, 2013) and reduce signal dropout rates (Jain & Das, 2005; Li & Cuthbert, 2004). In Virtual Circuit Routing and Admission control, disjoint-paths are often required when there is only limited bandwidth available (Awerbuch, Gawlick, Leighton, & Rabani, 1994)

As these problem types tend to be quite large, greedy based heuristics have traditionally been used to solve the MEDP problem. Examples of such algorithms include the Simple Greedy Algorithm (SGA) (Kleinberg, 1996), the bounded-length greedy algorithm (Kleinberg, 1996), and the greedy path algorithm (Kolliopoulos & Stein, 2004). In 2007 Blesa and Blum developed the first metaheuristic to solve the MEDP problem using an Ant Colony Optimisation (ACO) algorithm (Blesa & Blum, 2007). This ACO algorithm is still used as a benchmark for the MEDP problem. Since 2012, a variety of new methods have been developed, including a Constraint based Local Search (Pham, Deville, & Van Hentenryck, 2012), a Genetic Algorithm (Hsu & Cho, 2015), a Message Passing algorithm (Altarelli, Braunstein, Dall'Asta, De Bacco, & Franz, 2015) and a Two-Stage Hybrid Metaheuristic (Martín, Sánchez, Beltran-Royo, & Duarte, 2020). A limitation these existing techniques face is the inability to provide any performance guarantees in the context of optimality gaps. In this paper we attempt to outperform

* Corresponding author.
E-mail addresses: jake.weiner@rmit.edu.au (J. Weiner), andreas.ernst@monash.edu.au (A.T. Ernst), xiaodong.li@rmit.edu.au (X. Li), yuan.sun@rmit.edu.au (Y. Sun), kdeb@egr.msu.edu (K. Deb).

https://doi.org/10.1016/j.ejor.2021.01.009
0377-2217/

these previous methodologies in both primal solution qualities of the non-trivial, large problem instances, as well as being the first paper to provide good Upper Bounds for the instances tested. To accomplish both goals, we use a modified Lagrangian Heuristic and Particle Swarm Optimisation Hybrid (LaPSO) algorithm, originally developed by (Ernst, 2010; Ernst & Singh, 2012). Whilst this hybrid method has worked well on a limited number of problems, this paper provides a more extensive application test, carrying out further examination of the inner workings behind the LaPSO algorithm.

The motivation for using a Lagrangian Heuristic based technique arises after modelling the MEDP problem as an Integer Program (IP). By relaxing the edge capacity constraints for the MEDP problem, the problem can be decomposed into a series of independent shortest path problems, which are easily solvable using a well known shortest path algorithm such as Dijkstra's method (Dijkstra, 1959). This relaxation approach, termed Lagrangian Relaxation (LR) (Fisher, 1981), has to the best of our knowledge not been attempted previously to solve the MEDP problem. Solving the relaxed problem provides us with Upper Bounds for the optimal solutions to the original MEDP problem, something traditional heuristic based techniques are unable to provide. Whilst a traditional Mixed Integer Programming (MIP) solver could be used to provide bounds for small scale problems, these solvers are unable to do so in a reasonable amount of time as problem sizes increase. Solutions to the Lagrangian Relaxed problem are often almost feasible, and are suitable candidates for a good repair heuristic. Another unique contribution of this paper is the creation of a novel repair heuristic termed *Largest Violation Perturbation* (LVP) for the MEDP problem. The LVP repair heuristic uses *perturbation* variables which are introduced in the LaPSO algorithm to increase exploitation of promising solutions, something which traditional LR techniques lack.

In addition to the creation of a new repair heuristic, we provide further analysis surrounding the effect that the perturbation variables have on both the primal solution and bound qualities, something which has not been done previously (Ernst, 2010; Ernst & Singh, 2012). We also provide further tests surrounding the particle swarm, showing the effect a population based metaheuristic can have when hybridised with a traditional exact technique. The adapted LaPSO method was found to produce significantly better results, becoming the new state-of-the-art for many of the problem instances tested, particularly on the larger problem instances. We also show that LaPSO can generate tighter bounds than a traditional MIP solver and the LR based Volume algorithm (Barahona & Anbil, 2000), especially for the largest instances tested when given a limited CPU runtime. This demonstrates that our hybrid method outperforms both base optimisation approaches (metaheuristics and Lagrangian Relaxation) from which it draws inspiration.

The rest of this paper is structured as follows. Section 2 introduces the background and related work. Our main approach is described in Section 3. Section 4 details the experimental design and presents the results. Section 5 then concludes the paper.

2. Background and related work

In this section we summarise the MEDP problem, the previous methods used to solve the MEDP problem as well as provide the necessary background required to understand the LaPSO algorithm. The LaPSO algorithm incorporates techniques such as: Lagrangian Relaxation, Wedelin's Algorithm and Particle Swarm Optimisation.

2.1. Maximum edge disjoint problem

The MEDP problem can be defined as follows: Let $G = (V, E)$ represent an undirected graph. Let $K = \{s_j, t_j\} | j = 1 \dots |K|$ represent a list of commodities K containing both source and terminal nodes (s_j, t_j), for which the commodity must be transported from and to respectively. Each edge $e \in E$ has a capacity constraint of one, meaning it can transport at most one commodity. The objective of the MEDP is to maximise the number of source and terminal node pairs that can be successfully connected whilst also ensuring that each edge does not violate its capacity constraint. An example of the MEDP problem can be seen in Fig. 1. In the MEDP problem we consider in this paper, we assume all edge costs to be zero, and therefore all paths, no matter their length, are considered equally as good. Determining whether a set of commodities can successfully be connected in this way is famously one of Karp's original NP-hard problems (Karp, 1972).

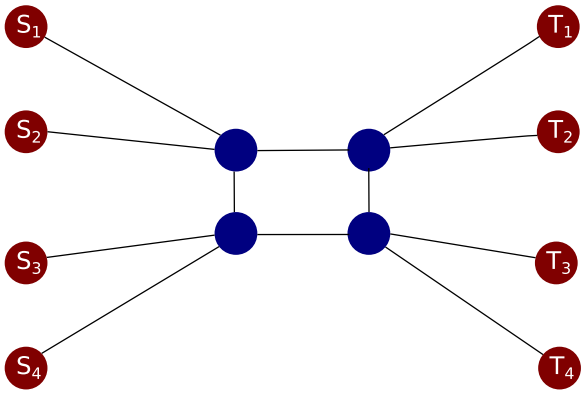


Fig. 1. An example of the MEDP problem. In the MEDP problem, the complicating constraint is that each edge in the network has a capacity constraint of one. In this example, where there exist 4 commodities to be routed across the network from their respective source nodes (S_i) to their respective terminal nodes (T_i), due to the edge capacity constraints, the maximum number of commodities that could be successfully routed through the network from their source to terminal nodes is two.

2.2. Previous MEDP attempts

For a long time the only type of methods used to solve the MEDP had been simple greedy heuristics. Then Blesa and Blum in (Blesa & Blum, 2007) described the first metaheuristic based approach to solve the MEDP problem. Since then, a variety of techniques have been tested, including Constraint-Based Local Search (Pham et al., 2012), Genetic Algorithm (Hsu & Cho, 2015), Message Passing (Altarelli et al., 2015) and a Two-stage Method (Martín et al., 2020).

2.2.1. Greedy heuristics

The MEDP problem is usually required to be solved on large graphs, and as a result many greedy heuristic based algorithms were originally explored. Greedy heuristic algorithms tend to be fast, however they can produce results which are quite far from being optimal. The greedy heuristic algorithm displaying the best time-complexity is known as the Simple Greedy Algorithm (SGA) (Kleinberg, 1996) and is commonly used as a benchmark for the MEDP problem (Blesa & Blum, 2007). The SGA algorithm simply iterates through all commodity pairs, finding the shortest path between source and terminal nodes using available edges in the network. When an edge is used in a path, it is discarded from the list of available edges, and is no longer accessible for other commodity paths. As discussed in (Blesa & Blum, 2007), finding the optimal solution using this approach is sometimes not possible due to the deterministic decisions made during the solution construction phase. Improving on the SGA can be done through implementing multiple restarts, and has been labelled as the Multiple Start Greedy Algorithm (MSGA) (Blesa & Blum, 2007). This

approach builds upon the SGA by allowing for multiple restarts thereby ensuring that different iteration orders are checked, reducing the problem of determinism that the SGA faced.

2.2.2. ACO

Blesa and Blum's Ant Colony Optimisation (ACO) approach (Blesa & Blum, 2007) represents the first metaheuristic used to solve the MEDP problem. The ACO approach decomposes the MEDP problem into $|K|$ sub-problems, where $|K|$ is the number of commodities. Each sub-problem is tasked with finding a suitable path connecting its respective source and terminal node pair. Each sub-problem is solved by a different "ant" using a pheromone model which is common in all ACO algorithms. Whilst the solution produced by all $|K|$ ants may not be feasible, it can be repaired using a simple repair heuristic. In this case, the repair heuristic consists of iteratively removing paths, in order of those containing the most number of shared edges until a feasible solution is found (Blesa & Blum, 2007). Whilst the ACO method was state-of-the-art for a long time, recently two other methods have been developed which represent the new state-of-the-art.

2.2.3. Constraint-Based local search

The Constraint-Based Local Search method introduced in (Pham et al., 2012) has two variants - LS-SGA and LS-R. LS-SGA initialises a solution by randomly creating paths for all commodity source-terminal pairings. It then applies a "LocalMove" operator in an effort to decrease the number of edge violations. A repair heuristic called Extract is applied which iteratively removes commodities until a feasible solution exists. Edges from these solutions are removed from the graph, and a Simple Greedy Algorithm (SGA) is applied to the remaining edges for each commodity pair not connected. LS-R relies on recursive calls, each time storing the feasible paths that are found when connecting commodity source-terminal pairs. The edges involved in these solutions are removed from the graph during each recursive call. A Greedy Local Search is used to create paths connecting each commodity's source and terminal nodes.

2.2.4. Genetic algorithm

Hsu and Cho developed a Genetic Algorithm (GA) to solve the MEDP problem (Hsu & Cho, 2015). Each individual within the population contains $|I|$ paths, where each path is encoded with real values in the interval [0,1]. Each of these values represents the nodes priority of being selected during path construction. Given a pair of source and terminal nodes, s_k and t_k, starting from s_k a path is created by considering nodes adjacent to s_k, with the highest priority node being selected. Such a procedure is continued until the terminal node t_k is reached. This procedure is carried out k times, creating k paths. A repair heuristic is then run, keeping only a subset of these paths such that there is no interference between the paths. Three genetic operators: crossover, mutation and a self-adaptive operator are used to create diversity in the population and form new offspring. The crossover operator uses a linear combination of two individuals to form an offspring. The mutation operator reverses the priority values of randomly selected commodities, transforming large priorities into smaller ones and vice versa. Finally the self adaptation operator randomly selects a path from an individual and tries to re-route a disconnected commodity by assigning new priority values to the nodes. The new priority scores are calculated using features such as path length and edge usage rates.

2.2.5. Message passing

The Message Passing algorithm (Mézard & Parisi, 2003) is a metaheuristic which has been applied to this problem in (Altarelli et al., 2015). It is based on the Cavity Method which has been used in other Combinatorial Optimisation problems. The Message Passing algorithm draws inspiration from an incremental learning method with weighted matching problems solved to compute updates. For interested readers we encourage them to see the original paper (Altarelli et al., 2015).

2.2.6. Two stage algorithm

The Two Stage Algorithm as described in (Martín et al., 2020) consists of an Integer Linear Programming (ILP) solver being used to generate seed solutions for an Evolutionary Algorithm (EA). If the optimal solution is produced by the ILP solver, the process is terminated and the solution is reported. Each individual within the EA population is represented by an array of size K, equaling the number of commodities in the problem instances. For each commodity $k \in K$, λ shortest paths are generated between the source-terminal node pair. To generate a new offspring, a crossover operator compares the current individual, the local best solution of that individual, and the global best solution amongst the population and chooses the gene from one of these candidates using a roulette wheel strategy. A post-processing stage then takes places, as well as the updating of objective values for the individual, local best and global best solutions.

2.3. LaPSO background

The Lagrangian Particle Swarm Optimisation (LaPSO) method was first described in (Ernst, 2010; Ernst & Singh, 2012). The LaPSO algorithm consists of a Wedelin inspired Lagrangian Relaxation (LR) method embedded within a Particle Swarm Optimisation (PSO) framework. This approach has proven to be successful in other combinatorial optimisation problems such as the Resource Constrained Machine Scheduling Problem (Ernst & Singh, 2012), as well as the Degree-Constrained Minimum Spanning Tree Problem (Ernst, 2010). In order to explain the workings of the LaPSO algorithm, we give a brief background on Lagrangian Relaxation, Wedelin's Algorithm and Particle Swarm Optimisation. For the full pseudocode please refer to Algorithm 1.

2.3.1. Lagrangian relaxation

The inner method used in the LaPSO algorithm, known as Lagrangian Relaxation (LR), is an exact technique that is commonly implemented in the Operations Research (OR) community. LR arose from the observation that some hard problems can be modelled as relatively easy problems with complicating constraints. For problems of this nature, if these constraints were removed, the problem would be much easier to solve.

The basic form of the primal and Lagrangian dual problems for Binary Linear Programs are shown in Eq. (1) and (2) respectively, where x are the decision variables, c are the associated costs, A and b are the constraint matrix and resource constraints and λ are the Lagrangian Multipliers introduced.

$$\max_{x} \quad f(x) = c^T x \quad \text{s.t.} \quad Ax \leq b, \quad x \in \{0,1\}^n \tag{1}$$

$$\min_{\lambda \geq 0} LR(\lambda) = \max_{x \in \{0,1\}^n} c^T x + \lambda^T (b - Ax) \tag{2}$$

Note that this can be extended in a straight forward way to the case of Mixed Integer Programming. LR is carried out by relaxing some of the constraints (or as presented here all of the constraints) and shifting them to the objective function, with some penalty term (Lagrangian Multiplier) attached. Solving the new relaxed problem provides bounds for the solution qualities achievable in the original problem. In many cases finding the optimal Lagrangian Multipliers and the tightest bounds are the main objective when solving the relaxed problem. However, finding optimal Lagrangian Multipliers can also result in finding good feasible solutions, as LR is essentially a penalty method.

Algorithm 1 MEDP LaPSO Algorithm.

Require: Number of particles n
Require: CPU Runtime Limit T
Require: Velocity parameters $\omega, \phi, \delta > 0$
Require: α^0, β the initial value and update factor of α
Define global best $\lambda^G = 0$, $p^G = 0$
Set initial perturbation and Lagrangian Multipliers for $p_i(\lambda_i, \pi_i) \in n : \lambda_i \in [0, 0.1]$, $\pi_i = 0$
Set initial velocities $v_i = 0$, $w_i = 0$
Set maxIter and LBC $maxIter = 30000$, $LBC = 10$
for $iter = 0 \ldots maxIter$ **and** while $LB \leq UB - \epsilon$ **do** and while $CPU() \leq T$
  for $p_i(\lambda_i, \pi_i) \in n$ **do**
    if $iter \mod LBC = 0$ **then** // Lower Bound check
      Solve subproblem $L(\lambda_i, 0)$ // using Dijkstra
    else
      Solve subproblem $L(\lambda_i, \pi_i)$ // using Dijkstra
    end if
    Let x' be the optimal solution to theminimisation subproblem
    $g \leftarrow b - Ax'$ // update subgradient g
    if $L(\lambda_i, \pi_i) > LB$ **then**
      $LB \leftarrow L(\lambda_i, \pi_i)$
    else if LB_i not improved for 10 iterations of particle p_i **then**
      $\alpha_i \leftarrow \beta \alpha_i$ // reduce step size
    end if
    if x' is not feasible **then**
      $x' \leftarrow$ RepairHeuristic(x') // repair solution
    end if
    if x' is feasible and $(\bar{c}x' < UB)$ **then**
      $UB \leftarrow \bar{c}x'$ // update best Upper Bound
      $p^G \leftarrow p_i$ // update best particle
    end if
    Randomly generate numbers $r^G, r^L \in [0, 1]$
    Update particle velocities v_i, w_i using Eq. (13) and (14)
    Update the particle positions using Eq. (15) and (16)
  end for
end for

Whilst there are many techniques available to find the optimal Lagrangian Multipliers, traditionally the most popular method has been sub-gradient optimisation (Fisher, 1981). The Lagrangian function is non-smooth. Hence while a gradient may not exist, the function always has a set of *subgradients* corresponding to the normal vectors of a set of supporting hyperplanes at a point on the function's level set. Subgradient optimisation uses an arbitrarily chosen subgradient of the Lagrangian function with respect to the current Lagrangian Multipliers, as a guide to update the multipliers. A subgradient is given by the constraint violation, as can be seen in Eq. (2), with alternative subgradients corresponding to different optimal subproblem solutions when there are multiple optima. Following the direction of the subgradient modifies the penalties such that it encourages more feasible solutions. Penalties are increased where the constraints are violated, and decreased where there is slack in the constraint. During this update process, feasible solutions for the original problem may occur on their own, or most likely, "almost feasible" solutions can be found. A repair heuristic is often then applied to repair these "almost feasible" solutions, with this combination of LR and repair heuristic commonly referred to as a Lagrangian heuristic.

2.3.2. Wedelin's algorithm

Because subgradient optimisation operates solely in the relaxed problem space, there may exist situations such that even if the optimal Lagrangian Multipliers are found, the solutions generated using these variables may never be feasible in the original search space (Ernst, 2010; Ernst & Singh, 2012). To tackle this problem of potential infeasibility, the LaPSO algorithm loosely incorporates ideas explored in Wedelin's algorithm (Wedelin, 1995), which was originally developed for 0–1 combinatorial optimisation problems. Wedelin's algorithm does not use a subgradient method to update the Lagrangian Multipliers. Instead, Wedelin's algorithm updates the Lagrangian Multipliers based on a coordinate ascent approach, iteratively adjusting each Lagrangian Multiplier λ_i such that the solution x exactly satisfies the i^{th} relaxed constraint of $Ax \leq b$. It does so by reformulating the LR in terms of the adjusted costs $\hat{c} = (c - \lambda A)$ and ensuring that at most b elements involved in constraint i have negative reduced costs (assuming coefficients of A are 0–1). The solver can then easily choose which elements to set to 1, and which to set to 0. This approach can work well for small problem instances, however for more difficult problems many adjusted costs converge to 0, and as a result no integer solution is found. To combat this, Wedelin also provides an approximation version, with perturbation values as well as Lagrangian Multipliers affecting the adjusted variable costs. After each multiplier update, if the adjusted variable cost was positive, a small positive perturbation is added, and likewise a small negative value is added if the adjusted cost was negative. Over time, these perturbations help to break ties amongst the adjusted costs which would otherwise converge to 0. In a similar manner, the LaPSO algorithm biases variables in an effort to break ties. It does so using previous best global solutions as the guiding mechanism. No attempt has been previously carried out to use this approach on the MEDP problem. How this is implemented in the context of the MEDP problem is explained in more detail in Section 3.2.

Whilst the perturbation values are used in an effort to create more feasible solutions, often without a repair heuristic the solutions created will still be infeasible. Hence Lagrangian methods rely on *repair heuristics* that modify an infeasible solution to the relaxed subproblem, hopefully finding a similar solution which is feasible for the original problem. A good repair heuristic is often crucial to the performance of a Lagrangian heuristic. The repair heuristic generated for the MEDP problem is discussed in Section 3.3.

2.3.3. Particle swarm optimisation

Particle Swarm Optimisation (PSO) is a longstanding metaheuristic originally presented in (Eberhart & Kennedy, 1995). Like many other metaheuristics, PSO draws its inspiration from nature, mimicking swarms found in nature, such as bird flocks. In most traditional PSO algorithms, particle velocity and position updates are carried out using a combination of local and global best solutions. In this sense the velocity and position updates of each particle in LaPSO is carried out in a similar manner as described in (Shi & Eberhart, 1998). Unlike the more typical PSO methods, the LaPSO algorithm assigns two velocities and positions to each particle, one for the Lagrangian Multiplier space and one for the perturbation space, further discussed in Section 3.4. From a theoretical perspective, the hybridisation of a Lagrangian Relaxation approach and the PSO algorithm is based on the previous success of bundle methods (Hiriart-Urruty & Lemaréchal, 1993), which have been shown to be more effective than the subgradient optimisation algorithm for finding the optimal Lagrangian Multipliers (Zhao & Luh, 2002). Using the PSO algorithm, we are effectively relying on multiple subgradients to update the Lagrangian Multipliers, in particular the Lagrangian Multipliers found in the global best particle. This in theory should help avoid the oscillations between the different hyperplanes in the Lagrangian Relaxation search space which can often occur when using the subgradient optimisation technique (Zhao & Luh, 2002). As for the choice of using the PSO alogorithm as opposed to other metaheuristic techniques, the PSO algorithm has previously shown fast convergence properties for continuous optimisation problems (Poli, Kennedy, & Blackwell, 2007), which is essentially what the Lagrangian search space is.

3. Approach

In this section we provide details regarding the implementation of the LaPSO algorithm for the MEDP problem. This includes

reformulating the MEDP problem as an Integer Programming (IP) problem, applying a suitable Lagrangian Relaxation, creating a new repair heuristic, and embedding this all into the LaPSO framework. The MEDP LaPSO algorithm as seen in Algorithm 1 can be summarised as follows:

1. Relax capacity constraints, add perturbations and reformulate the problem as a minimisation problem instead of a maximisation one such that the MEDP problem is now a series of shortest path problems;
2. Solve the shortest path problems using Dijkstra's algorithm with a Priority Queue. Every t iterations temporarily set perturbations to 0 in an effort to improve upon the Lower Bounds;
3. If solutions generated are infeasible, apply a repair heuristic;
4. Using the solutions and constraint violations found in solving the subproblems, update the Lagrangian and perturbation values within the PSO framework.

It is important to note that for the LaPSO algorithm shown in Algorithm 1, the sub-problem is a minimisation problem and therefore the Lower Bounds (LB) represent solutions for the relaxed problem whilst the Upper Bounds (UB) represent feasible solutions for the original MEDP problem. This is in contrast with the bounds discussed in Section 4 which are referring to the original MEDP formulation which is a maximisation problem. This discrepancy arises because we transform the MEDP problem into a minimisation problem in the LaPSO framework, however, we present our results in the context of the original maximisation problem.

3.1. IP Formulation and Lagrangian Relaxation

In order to apply a Lagrangian Relaxation to the MEDP problem, we need to reformulate it as an IP problem as shown in Eq's. (3) - (7) formulation. In this formulation S represents the set of source nodes, T is the set of terminal nodes, K is the set of commodities and E is the set of all edges within the network. The MEDP problem is similar to a network flow problem with flow constraints shown in (4) - (5), capacity constraints (6) and binary constraints (7).

Note that, while this could in principle be solved by a generic Mixed Integer Programming (MIP) solver, these tend to perform badly because it is an NP hard problem and the formulations get quite large in both the number of variables and constraints ($|E| \times |K|$ and $|V| \times |K|$). Results generated using a generic MIP solver are shown in Section 4.

$$\max \sum_{k\in K} \sum_{j:(s_k,j)\in E} x_{s_k jk} \tag{3}$$

$$\sum_{j:(i,j)\in E} x_{ijk} - \sum_{j:(j,i)\in E} x_{jik} = 0 \qquad \forall\ k \in K, i \in V \setminus \{s_k, t_k\} \tag{4}$$

$$\sum_{j:(s_k,j)\in E} x_{s_k jk} \le 1 \qquad \forall\ k \in K \tag{5}$$

$$\sum_{k\in K} (x_{ijk} + x_{jik}) \le 1 \qquad \forall\ (i,j) \in E : i < j \tag{6}$$

$$x_{ijk} = \{0, 1\} \qquad (i,j) \in E, k \in K \tag{7}$$

We then relax capacity constraints (6) using Lagrangian Multipliers $\lambda \ge 0$, to obtain the following Lagrangian objective, where $\mathcal{F}$ is the set of feasible solutions to constraints (4), (5) and (7) (and $\mathcal{F}_k$ the corresponding subspace for commodity $k \in K$ since $\mathcal{F}$ is separable by k). For the following, let $\bar{E} = \{(i,j) \in E : i < j\}$.

$$\min_{\lambda\ge 0} LR(\lambda) = \max_{x\in\mathcal{F}} \sum_{k\in K} \sum_{(s_k,j)\in E} x_{s_k jk} + \sum_{(i,j)\in\bar{E}} \lambda_{ij}\left(1 - \sum_{k\in K} (x_{ijk} + x_{jik})\right) \tag{8}$$

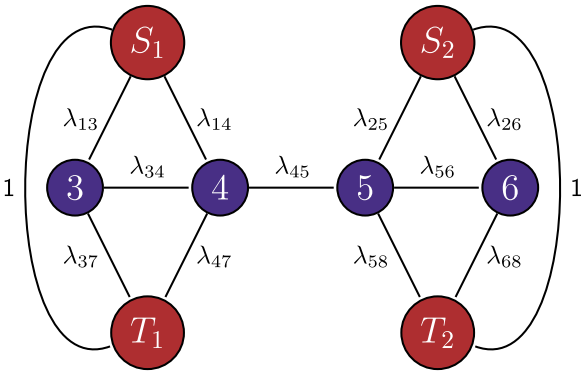


Fig. 2. A visual represenation of the MEDP problem with LR applied and non-negative edge weights as decribed in (11). This reformulated problem represents a series of independent shortest path problems for each Source/Terminal (S_i, T_i) node pairing with the Lagrangian Multipliers forming the edge weights. As described in (11), if the shortest path for a commodity is > 1, then the commodity is not routed through the network.

$$= \sum_{(i,j)\in\bar{E}} \lambda_{ij} + \max_{x\in\mathcal{F}} \left\{ \sum_{(s_k,j)\in E} x_{s_k jk}(1 - \lambda_{s_k j}) - \sum_{k\in K} \sum_{(i,j)\in\bar{E}, i\ne s_k} \lambda_{ij} x_{ijk} \right\} \tag{9}$$

$$= \sum_{(i,j)\in\bar{E}} \lambda_{ij} - \sum_{k\in K} \left\{ \min_{x_k\in\mathcal{F}_k} \sum_{(s_k,j)\in E} (\lambda_{s_k j} - 1) x_{s_k jk} + \sum_{(i,j)\in\bar{E}, i\ne s_k} \lambda_{ij} x_{ijk} \right\} \tag{10}$$

The last step follows as each of the path problems for $k \in K$ are independent and minimising the negative of the cost is the same as maximising. Now for each commodity k we essentially have two choices: not routing the commodity through the network (in which case $x_{ijk} = 0$ for all $(i,j) \in E$) or alternatively, if a path is selected for some commodity k then there is a contribution of one to the LR value (from the fact that $\sum_j x_{s_k jk} = 1$) and we subtract the shortest path costs with respect to the λ_{ij}'s.

This demonstrates that the Lagrangian function can be computed using as a series of shortest path calculations using the equation

$$\min_{\lambda\ge 0} LR(\lambda) = \sum_{(i,j)\in E} \lambda_{ij} + |K| - \sum_{k\in K} \min\left\{1, \min_{s_k - t_k \text{path} P_k} \sum_{(i,j)\in P_k} \lambda_{ij}\right) \tag{11}$$

Here the inner minimisation is simply to find a shortest path in the undirected path between source-terminal pair (s_k, t_k) using edge costs given by the Lagrange multiplier λ. This new reformulation can be seen visually in Fig. 2. Since $\lambda \ge 0$ these shortest path problems can easily be solved for example with Dijkstra's algorithm, without needing to cater for the posibility of negative cost cycles. It is also obvious that while $LR(\lambda)$ calculates an upper bound for our maximisation problem, the calculation is effectively equivalent to finding a lower bound on the problem of minimising the number of commodities for which no arc disjoint path can be found.

3.2. Perturbations

When solving the series of shortest paths subproblems, within the network there may exist multiple paths connecting the same node pair with similar lengths as shown in Fig. 3. Despite paths having similar lengths, when solving the shortest path subproblem using Dijkstra's algorithm, all commodities sent between a node

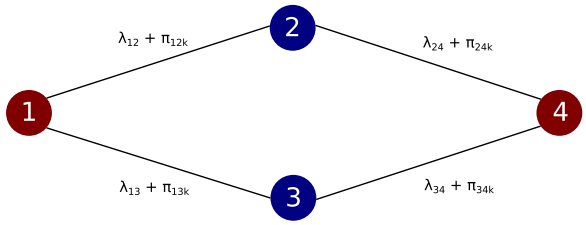


Fig. 3. An example showing the usefulness of the perturbation variables. Each commodity $k \in K$ has it's own unique perturbation value for each edge, biasing commodities to use specific edges and routes which were found in good historical solutions. This enables different commodities to take different routes which would otherwise not be possible using only the Lagrangian Multipliers as edge weights. In this example, without any perturbation variables, all commodities routed through nodes $1 \rightarrow 4$ would use the same edges as a result of solving the shortest path problem. This choice of path necessitates that all solutions would be infeasible, if multiple commodities are routed through this section of the network. Using perturbation values, if alternative edges with similar costs (Lagrangian Multpliers) are available, these perturbation variables are used to bias commodities into using these alternative edges if historically this resulted in producing a good solution.

pair will traverse the same shortest path, resulting in edge violations and infeasible solutions. To avoid this, we introduce perturbation variables in a similar manner to Wedelin's algorithm as described in Section 2.3.2. These perturbation variables bias commodities to be routed along unique paths, helping to avoid infeasible solutions. Note that these perturbation values are small in relation to the Lagrangian Multipliers as to not negatively affect the Bound quality and the Lagrangian Multiplier updates.

We assign a small perturbation variable to each commodity-edge pairing, π_{ijk}, as shown in Fig. 3. Referring to Fig. 3, let Path 1 be represented by edges {(1,2),(2,4)} and Path 2 by edges {(1,3),(3,4)}. If values $\lambda_{12} + \lambda_{24}$ are similar, but slightly smaller than $\lambda_{13} + \lambda_{34}$, all commodities routed through this section of the network will always be routed across Path 1, leading to infeasible solutions. By changing the perturbation values commodity-edge pairing, we can bias commodities to take different paths than they would if only the Lagrangian Multipliers were used as edge weights. These perturbation values are updated using historical solutions as a guide. Previous solutions provide insight surrounding the relationship between commodities using particular edges and overall solution quality. This can be used to bias commodities to favour particular edges, where previously this has lead to good quality solutions. Some randomness is also included in the update procedure, to create a balance between exploration and exploitation. The perturbation values are updated using information from the current globally best feasible solution, as shown in Eq. (14) in Section 3.4.

After introducing these perturbation variables, we reformulate the relaxed objective function to that as shown in Eq. (12). We then apply the same process described in Section 3.1 to transform this problem into a series of independent shortest path problems. Because we have added in these artificial perturbation values, our new objective function does not give us the true bounds for the original problem. We would have to subtract all perturbation values which are used in the paths generated. Instead of keeping track of all perturbation values used, we simply assume the worst case scenario and subtract the maximum perturbation value n times, where n represents the number of edges in the problem.

$$\begin{aligned}\min_{\lambda \geq 0} LR(\lambda, \pi) = \max_{x} & \sum_{k \in K} \sum_{(s_k, j) \in E} x_{s_k jk}(1 - \lambda_{s_k j} - \pi_{s_k jk}) \\ & - \sum_{k \in K} \sum_{(i,j) \in E, i \notin s} (\lambda_{ij} + \pi_{ijk})(x_{ijk} + x_{jik}) \\ & - n \max_{ijk}(\pi_{ijk}) + \sum_{(i,j) \in E} \lambda_{ij} \end{aligned} \tag{12}$$

3.3. Repair heuristic

When repairing solutions in the MEDP problem, we can iteratively remove routed commodities until a feasible solution is found. At the same time a commodity's path is removed, we try to find an alternative path for the commodity, using only available edges. There are therefore two decisions to be made using this approach: (1) In what order should commodities be removed in and (2) How do we find new paths for these commodities. We investigated three different repair heuristics for the MEDP problem. We have labelled these methods *Random* (*Rand*), *Largest Violation Arbitrary* (*LVA*) and *Largest Violation Perturbation* (*LVP*). For all heuristics, to try and find a new feasible path for a commodity, Dijkstra's algorithm is re-run using the edge weights as described in each heuristic.

3.3.1. Rand

In the first approach *Rand*, we randomly select which commodities should be removed. Once a commodity is removed, we try and find an alternative shortest path for the commodity using only available edges. We arbitrarily set the weight of each edge to one, meaning no edge is biased more than any other.

3.3.2. LVA - Largest Violation Arbitrary

In the second approach *LVA*, we iteratively remove commodities with the largest number of edge violations contained within the path. We then try and reconnect these commodities again using arbitrary edge weights assigned to available edges.

3.3.3. LVP - Largest Violation Perturbation

The final repair heuristic explored is *LVP* which iteratively removes commodities with the largest number of violations, and attempts to reconnect them using the perturbation values as edge weights. Again only edges which are available are considered. The potential advantages of using the perturbations to guide the repair heuristic is something which has previously not been explored. The perturbations used in the LaPSO algorithm are essentially learned values, influenced by past best solutions. This results in commodity paths converging to specific edges which were found to be highly beneficial in previous solutions. It results in more exploitation of promising solutions as the perturbation values are updated over time. Because perturbations can be negative, we have shifted all perturbations up to ensure no negative edge weights exist when solving our shortest path problems. Whilst the paths now solved do not represent true shortest paths, for our purposes this is acceptable.

3.4. Particle swarm optimization implementation

The velocity and position update procedure used in the LaPSO is as follows:

$$\boldsymbol{v}_i \leftarrow \omega \boldsymbol{v}_i + r^L \alpha_i \frac{UB - LB_i}{||\boldsymbol{g}_i||^2} \boldsymbol{g}_i + r^G \phi(\boldsymbol{\lambda}_i^G - \boldsymbol{\lambda}_i) \tag{13}$$

$$\boldsymbol{w}_i \leftarrow \omega \boldsymbol{w}_i + r^G \phi(\boldsymbol{\pi}_i^G - \boldsymbol{\pi}_i) + \delta r^G \phi(1 - 2\boldsymbol{x}_i^G) \tag{14}$$

$$\boldsymbol{\lambda}_i \leftarrow \boldsymbol{\lambda}_i + \boldsymbol{v}_i \tag{15}$$

$$\boldsymbol{\pi}_i \leftarrow \theta \boldsymbol{\pi}_i + \boldsymbol{w}_i \tag{16}$$

As mentioned previously, $\boldsymbol{\lambda}$ and $\boldsymbol{\pi}$ represent the dual and perturbation vectors, with $\boldsymbol{v}$ and $\boldsymbol{w}$ representing their velocities respectively. r^G and r^L both represent random variables uniformly distributed between [0,1]. ω is the velocity factor representing the

Table 1
Quantitative measures of experimental Dataset 1 (Blesa & Blum, 2007).

Graph	|**V**|	|**E**|	Min Deg.	Max Deg.	Ave Deg.	Diam.
graph3	164	370	1	13	4.51	16
graph4	434	981	1	20	4.52	22
AS-BA.R-Wax.v100e190	100	190	2	7	3.8	11
AS-BA.R-Wax.v100e217	100	217	2	8	4.34	13
bl-wr2-wht2.10–50.rand1	500	1020	2	13	4.08	23
bl-wr2-wht2.10–50.rand2	500	1020	2	11	4.08	27
bl-wr2-wht2.10–50.sdeg	500	1020	2	14	4.08	28
mesh15x15	225	420	2	4	3.73	28
mesh25x25	625	1200	2	4	3.84	48

inertia of each particle. ϕ is the global factor, and influences the effect that the global best solution has on each velocity. θ updates the perturbation values to ensure they do not grow too large, and is set to 0.5 as recommended in (Wedelin, 1995).

3.4.1. Lagrangian search space

The Lagrangian Multipliers are updated with a local component (subgradient optimisation) and a global component. The local component is simply a standard subgradient update step with $\boldsymbol{g}$ representing the subgradient of the LR function (the amount of constraint violation $b - Ax$), UB and LB representing the globally best Upper Bound and the locally best Lower Bound respectively. α_i is the Subgradient Multiplier, which is initialised as a parameter and is halved every 10 consecutive iterations where the LB is not improved. $\boldsymbol{\lambda_i}^G$ are the Lagrangian Multipliers used in the global best LB solution.

3.4.2. Perturbation search space

The perturbation update procedure is influenced through only the global solutions. $\boldsymbol{\pi}_i^G$ are the perturbation values belonging to the global best solution. The $(1 - 2\boldsymbol{x}_i^G)$ term adds a small positive bias to variables which are 0 in the global best solution, and adds a small negative bias to variables which are 1 in the global best solution. This biasing helps to ensure different commodities have different costs associated with each edge, increasing the chance that commodities do not share the same edge when routed through the network. Whilst the perturbation search space is larger than the Lagrangian search space, any feasible solution for the original problem can be found, something which would not be possible if only the smaller Lagrangian search space is relied on. δ is the Perturbation Factor which controls the size of the perturbations.

4. Experiments and results

Numerous experiments were designed to address: parameter tuning, perturbation analysis, comparative Upper Bound methods, repair heuristic selection and state-of-the-art benchmark comparisons.

The experiments were carried out using the Datasets described in (Blesa & Blum, 2007), (Pham et al., 2012) and our synthetically generated instances. We refer to these Datasets as Set 1, Set 2 and Set 3 respectively. A summary of these datasets can be seen in Table 1, Table 2 and Table 3 respectively. For each graph in Set 1, there exists 20 different instances for 0.1 $|V|$, 0.25 $|V|$ and 0.40 $|V|$ commodities totalling 540 problem instances. In Set 2, for graphs blrand, blsdeg, mesh15x15 and mesh25x25 there exist two different instances for 0.1 $|V|$, 0.25 $|V|$ and 0.40 $|V|$ commodities. For the rest of the graphs, there exist 1 instance for 0.1 $|V|$, 0.25 $|V|$ and 0.40 $|V|$ commodities. In total Set 2 contains 54 different problem instances.

For the newly generated instances, we use the pyrgg software tool as described in (Haghighi, 2017) to create what we have

Table 2
Quantitative measures of experimental Dataset 2 (Pham et al., 2012).

Graph	|**V**|	|**E**|	Ave Deg.
bl-wr2-wht2.10–50.rand	500	1020	4.08
bl-wr2-wht2.10–50.sdeg	500	1020	4.08
mesh15x15	225	420	3.73
mesh25x25	625	1200	3.84
steinb4	50	100	4.00
steinb10	75	150	4.00
steinb16	100	200	4.00
steinc6	500	1000	4.00
steinc11	500	2500	10.00
steinc16	500	12,500	50.00
planar-n50	50	135	5.4
planar-n100	100	285	5.7
planar-n200	200	583	5.83
planar-n500	500	1477	5.91

Table 3
Quantitative measures of experimental Dataset 3 .

Graph	|**V**|	|**E**|	Min Deg.	Max Deg.	Ave Deg.
easy_2_1	1000	3869	1	7	3.87
easy_2_2	1000	3921	2	6	3.92
easy_3_1	1500	5999	1	7	4.00
easy_3_2	1500	5984	2	6	3.99
easy_4_1	2000	7929	1	7	3.96
easy_4_2	2000	7962	2	6	3.98
easy_5_1	2500	10,048	1	7	4.02
easy_5_2	2500	10,114	2	6	4.05
Hard-small	500	1544	2	6	3.09
Hard-med	1000	3051	2	7	3.05
Hard-large	1500	4562	2	5	3.04

labelled “easy” and “hard” graph instances.[1] For the easy instances we attempted to recreate larger problem sizes with similar characteristics to the most difficult graphs found in Datasets 1 and 2. These difficult graphs were graph4, mesh25x25 and steinc6. The characteristics of these graphs displayed average degrees close to 4, which seemed to result in a reasonable level of difficulty when solving the MEDP problem. Whilst creating graphs with larger average degrees might result in larger problems in terms of both variable and constraint numbers, for path finding heuristic based techniques, these instances appear to be too easy to solve, with multiple paths available to produce optimal solutions. This can be seen in the results for the steinc16 instances in Table 6. We therefore created instances with average degrees close to 4, with varying minimum and maximum degrees. To test the limits of the LaPSO method, we created graph sizes for the “easy” instances containing between 1000 and 2500 vertices, far larger than any of the previous graphs tested. For each graph, we created 0.4 $|V|$ commodities, commonly referred to as Origin Destination (OD)

[1] These instances and the python source code used to generate them can be found at https://github.com/Jakew103/MEDP_EJOR.

pairs, in line with the most difficult instances found in Datasets 1 and 2.

To generate the "hard" graphs, we used a clustering approach in order to create instances in which some commodity OD pairs cannot be connected. To do so, we created 10 clusters of sizes 50, 100 and 150 vertices for the hard-small, hard-med and hard-large instances respectively. To make the instances more challenging, we also reduced the average degree for each graph, resulting in less paths available to transport each commodity. A single edge was created to link each cluster pairing, resulting in 45 inter-cluster edges in total. To generate the commodity OD pairs, we generated both intra-cluster and inter-cluster OD pairs. For each cluster, we generated $0.5|V|$ intra-cluster OD pairs, more than previous datasets which had been limited to $0.4|V|$. We then created 5 inter-cluster OD-pairs for each cluster pairing, knowing at most only 45 out of the 225 inter-cluster pairs could ever be successfully routed.

Parameter tuning and repair heuristic testing were carried out on Set 1, using one instance from each graph and commodity pairing, totalling 27 problem instances. These instances were run multiple times, providing a useful statistical analysis of the results. For perturbation analysis and comparative Lagrangian methods, all 540 instances from Set 1 were tested. For comparisons with the current state-of-the-art benchmarks, all 540 instances from Set 1 were used, as well as all 54 instances from Set 2. Each instance from Set 1 was run once, and each instance from Set 2 was run twenty times in accordance with the previous experiment designs as described in (Blesa & Blum, 2007; Pham et al., 2012).

Parameter tuning, repair heuristic selection and Upper Bound comparisons were carried out using a Intel(R) Core(TM) i7-7500U processor. The Mixed Integer Programming solver used for Upper Bound comparisons and heuristic solution benchmarking is CPLEX 12.8.0. CPLEX was run using the default solver settings. The LaPSO source code was written in C++ and compiled with gcc 5.4.0.[2] For Benchmark testing, the Multi-modal Australian ScienceS Imaging and Visualisation Environment (MASSIVE) network was used due to the large computational time required. Tests run on the MASSIVE network were done using either a Intel Xeon CPU E5-2680 v3 processor or an Intel Xeon Gold 6150 processor depending on which processor was available at the time. In all benchmark experiments for Datasets 1 and 2, runs were limited by the CPU times reported in either (Blesa & Blum, 2007) or (Pham et al., 2012), with the CPU time limit belonging to the method displaying the best average solution value in (Blesa & Blum, 2007) and 30 minutes per instance in (Pham et al., 2012) for consistency with the literature. This CPU limitation is important to keep in mind, as it affects the parameter values chosen. For the larger instances created in Dataset 3, the CPU runtime was limited to one hour per instance. It should also be noted that whilst the CPU time is a limiting factor amongst all algorithms, LaPSO is able to run on parallel CPU's, affecting wall clock runtimes. Whilst this measurement is not officially tested, it is an additional benefit of the LaPSO algorithm.

It is also important to note that in this section any references made to Lower Bounds are referring to primal solutions for the original MEDP problem, whilst Upper Bounds refer to solutions for the relaxed problem. In addition, any optimality gaps referenced are calculated as $100 \times (1 - \frac{LB}{UB})$.

4.1. Parameter tuning

We ran experiments to tune the five key parameters shown in Table 4 that significantly affect the performance of the LaPSO algorithm. Swarm size was first tested, ranging from a single particle to a swarm size of 96 particles. We tested swarms with multiples of 4 particles, simply because the LaPSO algorithm is implemented to run 4 particles in parallel across 4 CPUs. Shown in Fig. 4 are the convergence plots captured for the different swarm sizes. The test was carried out on one instance from each graph-commodity pair from Set 1, totalling 27 instances. As can be seen, the best results arise using a population of 8 particles. Looking at the convergence plots, the single particle suffers from premature convergence, with the average particle Lower Bound stagnating around a local optimum. On the other end of the spectrum, the 96 particle swarm also performs poorly. This is attributed to the large swarm only being able to complete fewer iterations. The limited number of iterations is a result of a fixed CPU time allowed for each instance in accordance with those reported in (Blesa & Blum, 2007). For some problem instances, when a larger swarm size was used, a single iteration was unable to be completed as the set-up time for LaPSO was longer than the CPU time limit. The plots shown in Fig. 4 are limited to 130 iterations to allow for easier comparisons, however it should be noted that the single particle was able to perform hundreds of iterations for some instances within the CPU time limit.

Table 4
Parameter tuning.

Parameter	Tuning Domain	Value Chosen
Velocity Factor - ω	{0.01,0.05,0.1,0.5}	0.1
Global Factor - ϕ	{0.01,0.05,0.1,0.2,0.3}	0.05
Subgradient Factor - α_i	{1.5,2,3,3.5,5}	2
Perturbation Factor - δ	{0.01,0.05,0.1,0.5}	0.5
Swarm Size - p	{1,8,16,32,64,96}	8

For the other 4 parameters - Velocity Factor, Global Factor, Subgradient Factor and Perturbation Factor, the iRace package (López-Ibáñez, Dubois-Lacoste, Cáceres, Birattari, & Stützle, 2016) was used for tuning. Again the same 27 instances were used for tuning. These instances vary in both problem size and complexity, avoiding any overtraining that might occur. Due to the number of instances, and the CPU time required by some of the large instances, only a limited domain was tested for the parameters. This domain was based around previously successful parameter choices as described in (Ernst, 2010; Ernst & Singh, 2012). The final elite-configuration given for the parameter choices is shown in Table 4.

4.2. Repair heuristic

The different repair heuristics, *Rand*, *LVA* and *LVP* were run 20 times each across a single instance from each graph-commodity pair from Set 1, totalling 27 instances. Multiple tests were run to account for the stochastic nature of the repair heuristics and LaPSO algorithm.

Comparing the three repair heuristics *Rand*, *LVA* and *LVP* as shown in Fig. 5, it is clear that the *LVP* heuristic produces the best results. We can see from the results that removing commodities based on their violations rather than in a random fashion, reduces the spread of results. This is to be expected, because we are removing a large random component from the heuristic. The biggest change however comes from using the perturbation variables as edge weights to guide the repair heuristic, as opposed to arbitrary values. This shows that the perturbation variables are converging to promising values which can be useful for such repair heuristics.

4.3. Perturbation analysis

We carried out experiments to understand better the effects of the perturbation values used, something which has not previously been done (Ernst, 2010; Ernst & Singh, 2012). To do so we

[2] The LaPSO source code can be found at https://github.com/Jakew103/MEDP_EJOR.

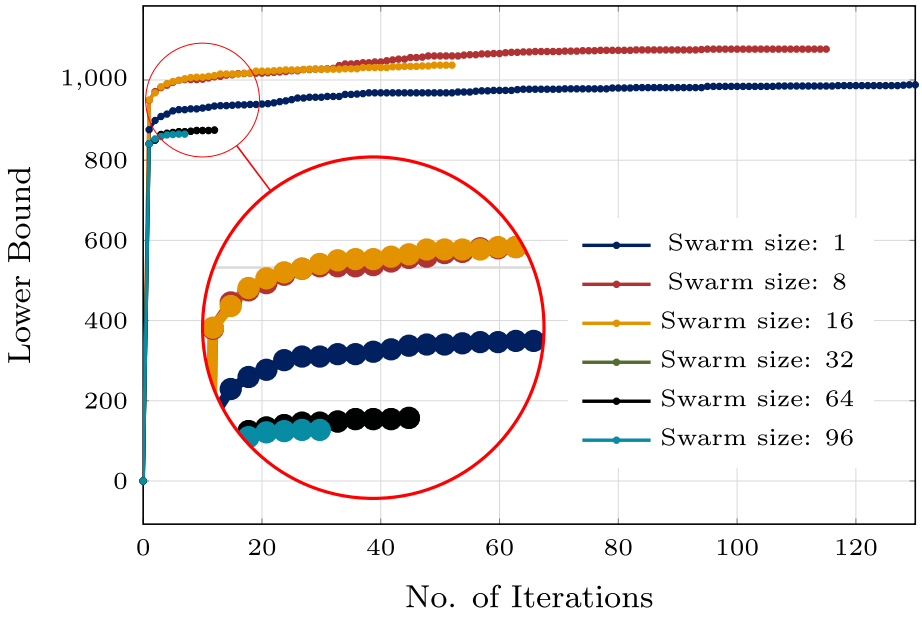


Fig. 4. Swarm Size behaviour analysis. Results from the 27 test problem instances, showing the sum of the best Lower Bound for different swarm sizes.

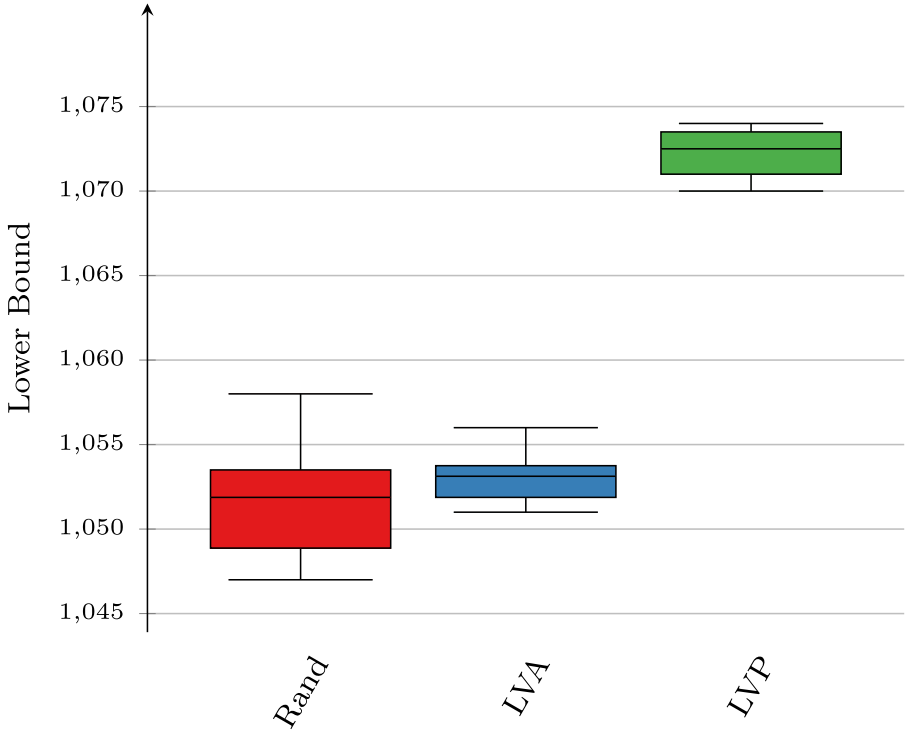
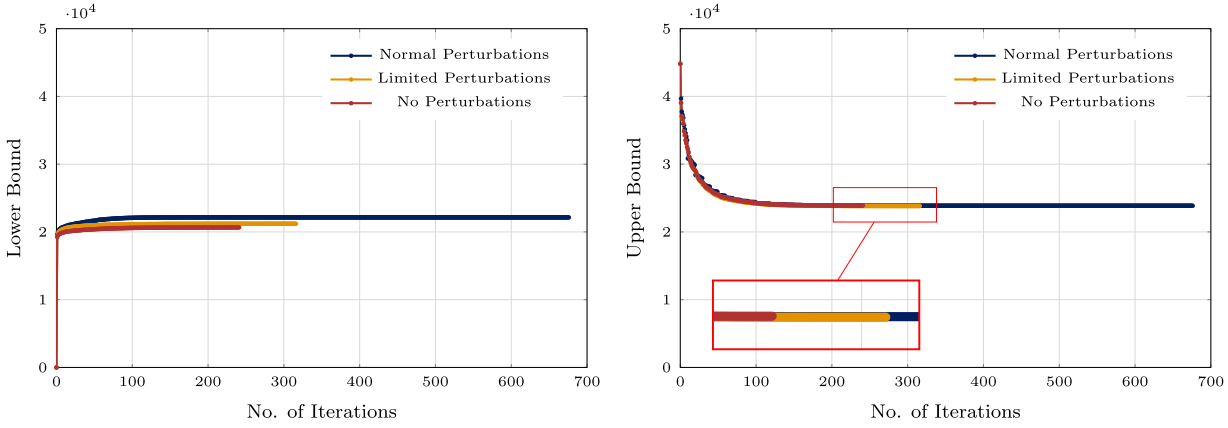


Fig. 5. Repair Heuristic comparisons. The results for each repair heuristic is shown in the form of a boxplot, using totals of the best Lower Bounds across all problem instances in each run. 540 problem instances were tested in total, running each of the 27 test problem instances 20 times.

tested all problem instances from Set 1 and Set 2 using the full LaPSO algorithm (Normal Perturbations), the LaPSO algorithm using the perturbation values only in the repair heuristic (Limited Perturbations) and the LaPSO algorithm using no perturbation values at all (No Perturbations). The results from these tests can be seen in Fig. 6. These tests allow us to see the effect of the perturbations in the subproblem solver by comparing Normal Perturbations and Limited Perturbations methods. We can also further see the effect of the perturbation variables in the repair heuristic by comparing the Limited Perturbations and No Perturbations methods. These experiments show that the perturbation variables are helpful in both the subproblem solver and repair heuristic without degrading the quality of the Upper Bounds generated. Whilst not necessarily the most important factor in performance, the average number of iterations able to be completed within the same runtime is influenced by the use of the perturbation variables. This is because the solutions generated are closer to being feasible and require less work to be carried out by the repair heuristic (fewer calls on average to Dijkstra's method to attempt to reroute commodities).



(a) LB Testing (b) UB Testing

Fig. 6. Perturbation Analysis comparing three methods - LaPSO with normal perturbation usage (Normal Perturbations), LaPSO without any perturbation values (No Perturbations) and LaPSO with using the perturbations only in the Repair Heuristic (Limited Perturbations). The graph on the left plots the sum of Lower Bounds against iteration number while the graph on the right plots the sum of the Upper Bounds for the same set of runs summed across all 540 instances tested.

4.4. Upper bound comparisons

Both a MIP solver (CPLEX) and the Lagrangian Relaxation based Volume algorithm were used to test the quality of the Upper Bounds that LaPSO is able to produce. In theory any MIP solver could be used to produce good quality Upper Bounds, however for large scale problems it is unable to do so in a reasonable amount of time. For the Maximum Edge Disjoint Path problem, because the integer solutions to the subproblems we have generated are the same as what can be achieved by a LP relaxation of the subproblems, the bounds produced by our Lagrangian based method will never be better than the optimal LP bounds. However, again for larger problem instances, the MIP solvers are most often unable to find the optimal solution, or even a good solution to the LP relaxation within a reasonable amount of time. To ensure that we obtain a valid upper bound even if the LP solver does not finish within the time limit, we used the dual simplex method. This is the default method used by CPLEX for these types of problems (when run with a single thread) and has the advantage that at every point during the running of the algorithm a feasible dual solution is available that provides a valid upper bound. We have also included the Volume algorithm (Barahona & Anbil, 2000) for comparison to compare the bounds generated by LaPSO with a more traditional Lagrangian based method. The implementation is based on the COIN-OR[3] implementation and the same approach for solving subproblems and heuristic repair method as in the LaPSO method.

We tested the five largest non-trivial problem instances available with 10 samples taken during each run at CPU times ranging from 90 seconds to 900 seconds. The instances tested include: bl-wr2-wht2.10–50.rand, bl-wr2-wht2.10–50.sdeg, mesh25x25, planar-n500 and steinc6. For the LaPSO algorithm, 30 runs were carried out to account for the stochastic properties of the algorithms. The results for these tests can be seen in Fig. 7. LaPSO clearly performed the best of the three methods tested, producing the tightest bounds in the limited CPU time available. We expect this trend to continue and be exaggerated even more as the problem sizes increase. As seen in Fig. 7, the dual simplex method is unable to make any significant progress for some of the instances, because the problems are extremely degenerate. In the worst performing example, the planar-n500 instance, we observed CPLEX spending the whole 900 seconds making over 140,000 successive degenerate pivots with no improvement to the bound from the value found with the initial basis. Similar extended periods of stagnation with over 100,000 degenerate pivots could be observed in most of the other test instances.

Given enough time, the MIP solver will eventually produce the tightest bounds once the optimal LP solution is found, however this is likely to occur only after larger runtimes. Its usefulness in future MEDP problem testing is therefore dictated by the available CPU time for the problem at hand.

4.5. Benchmark testing

For benchmark testing we ran the LaPSO algorithm using the parameter choices shown in Table 4 as well as the LVP repair heuristic. We ran the LaPSO algorithm and CPLEX solver once for each of the 540 problem instances in Set 1, and 20 times for each of the 54 problem instances in Set 2, in line with what was carried out in previous papers. Whilst this testing procedure was carried out in previous papers, for future work we recommend additional runs for Set 1 to account for the stochastic nature that is present in many of the benchmark algorithms.

For Datasets 1 and 2, we have split up the analysis into two parts. In part one LaPSO is compared with methods in the literature for which a reasonable comparison is able be carried out. These include current benchmark methods MSGA, ACO, LS-R and LS-SGA as described in (Blesa & Blum, 2007) and (Pham et al., 2012). We have also included a stand alone MIP solver, CPLEX 12.8.0 for benchmarking purposes.

Part two contains methods for which there is less confidence in the validity of the comparison, either because in the literature only limited test results could be found, or because the results were accompanied by incomplete information about how testing was carried out.

Finally, for our own synthetic Dataset created, we ran the LaPSO algorithm and CPLEX solver 20 times for the 11 newly created graphs and pairs files. We provide statistical measures, including both primal and dual solution benchmarks. We are hopeful that these newly created graphs will allow future researchers to test the effectiveness of their methods on harder instances than currently exist within the literature.

[3] Available from https://projects.coin-or.org/Vol.

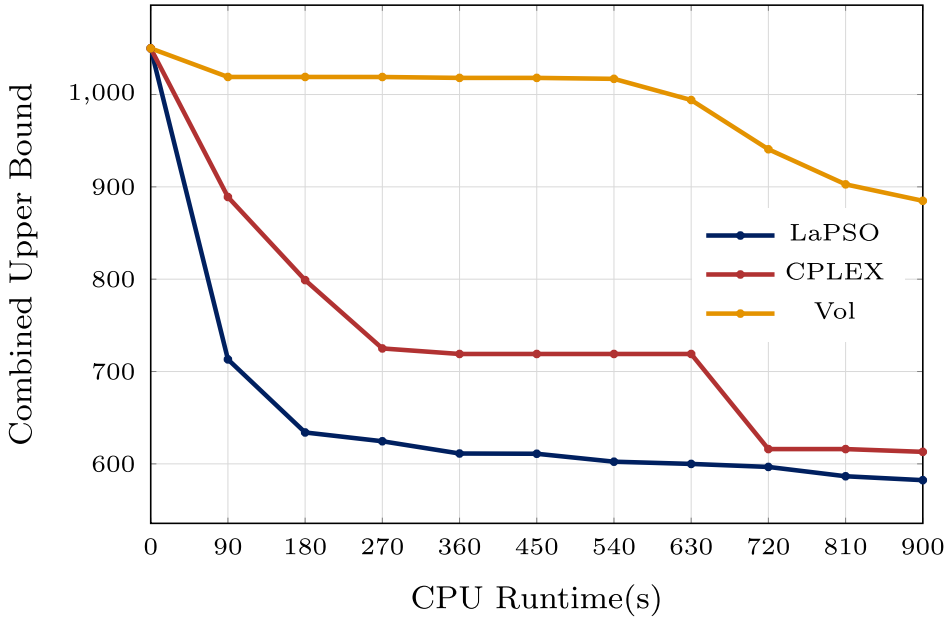


Fig. 7. Upper Bound Comparisons. Testing the five largest non-trivial instances from Datasets 1 and 2 using a MIP solver (CPLEX), LaPSO and a Volume algorithm (Vol). The Upper Bounds from each instance are summed up at 10 sample times, ranging from 90 to 900 CPU seconds. For LaPSO, 30 runs were completed to account for stochastic properties of the algorithm.

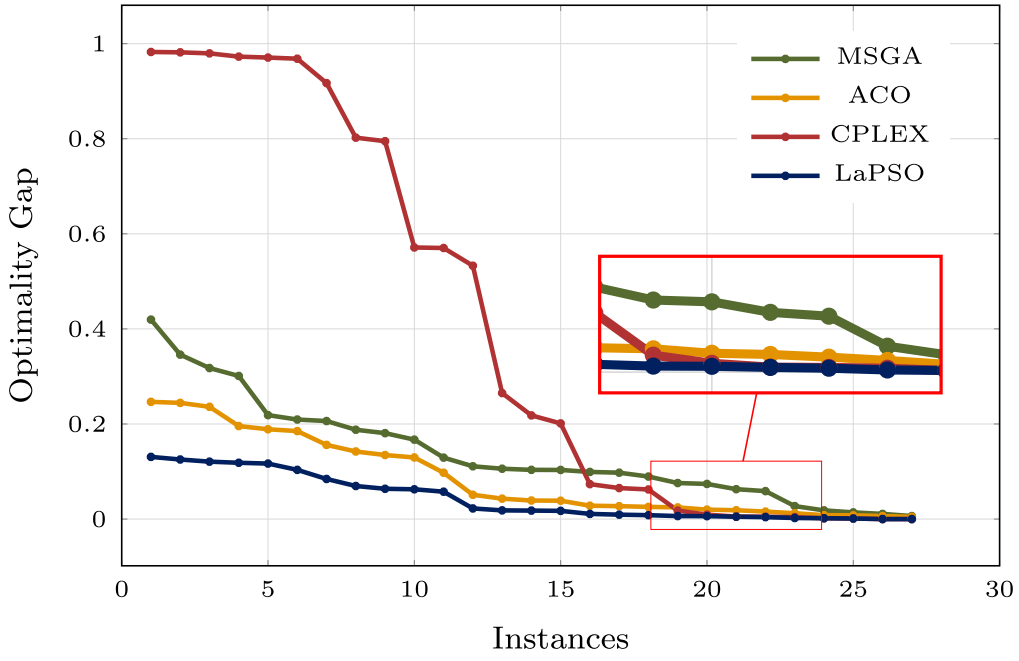


Fig. 8. Optimality Gap Comparisons. Optimality gaps calculated using the average Lower Bounds (from the heuristics) and Upper Bounds (from LaPSO) for the 20 different instances and each of the 27 graph-commodity pairs in Dataset 1. Optimality gaps are plotted in descending order for each method. The Instances axis does not refer to a specific instance, instead it is the index in the ordered sequence.

4.6. Set 1 - Comparable methods

LaPSO achieves outright state-of-the-art primal solution qualities for 16 out of the 27 commodity-graph pairs. Whilst the LaPSO method shows a higher average objective value across many of the problem instances, the small differences for the easier problems are not considered as significant, and could simply be attributed to the random nature that the different algorithms display. As the ACO method is generally superior to MSGA and CPLEX, a two tail *t*-test was carried out to compare the LaPSO and ACO methods. For the larger graphs such as graph4, mesh15x15 and mesh25x25, especially with the larger commodity instances, the LaPSO method significantly outperforms the ACO method. There is a statistically significant difference (p value $\leq$0.05) for 10 out of the 27 commodity-graph pairs tested. For the rest of the instances, most are almost able to be solved to optimal by both ACO and LaPSO, indicating these instances are relatively trivial to solve.

The LaPSO method is able to prove optimality on just over 47% of the problem instances. LaPSO achieves the lowest average optimality gap across all instances with 4.40%, compared with ACO, MSGA and CPLEX reporting average gaps of 8.63%, 13.86% and 40.64% respectively. A gap ranking comparison is shown in Fig. 8.

Table 5
Comparison results of the three methods MSGA, ACO, CPLEX and LaPSO for Dataset 1. The first column graph describes the graph structure used, with the ODs column describing the number of commodities to be transported for all runs from (Blesa & Blum, 2007). $\bar{q}$ represents the average objective value over the 20 different problem instances for each graph and commodity pairing. std represents the standard deviation of the objective values, whilst $\bar{t}$ represents the average CPU runtime taken to find the **best** solution within each instance. The best $\bar{q}$ are bolded to show the best average objective value between the MSGA, ACO, CPLEX and LaPSO methods. The Optimal column shows what proportion of instances were provably optimal when using the Upper Bounds found during the LaPSO algorithm. The final column "*t*-test" shows the two tailed t and p values used to compare the ACO and LaPSO methods.

Graph	ODs	MSGA			ACO			CPLEX			LaPSO					
		$\bar{q}$	std	$\bar{t}$	$\bar{q}$	std	$\bar{t}$	$\bar{q}$	std	$\bar{t}$	$\bar{q}$	std	UB	$\bar{t}$	Optimal	***t*-test (t, p)**
AS-BA.R-Wax.v100e190	10	9.10	0.94	0.58	8.95	0.97	0.61	8.60	2.18	2.29	9.10	0.94	9.20	0.02	0.90	(-0.58,5.68E-1)
AS-BA.R-Wax.v100e190	25	14.25	1.37	4.81	14.85	1.20	3.72	14.50	3.15	4.20	**15.30**	1.27	15.65	0.37	0.65	(-1.20,2.37E-1)
AS-BA.R-Wax.v100e190	40	17.95	1.62	7.80	19.45	1.94	4.12	16.85	7.10	5.37	**20.20**	2.11	21.55	1.37	0.20	(0.00,1.00E+0)
AS-BA.R-Wax.v100e217	10	8.05	0.92	0.43	7.88	0.93	0.16	6.55	3.04	2.32	8.05	0.92	8.20	0.02	0.85	(-0.68,5.02E-1)
AS-BA.R-Wax.v100e217	25	13.60	1.46	4.33	13.83	1.58	1.82	13.55	3.22	3.31	**14.20**	1.44	14.45	0.18	0.75	(-0.90,3.72E-1)
AS-BA.R-Wax.v100e217	40	17.00	1.95	9.83	17.80	1.65	2.21	15.25	6.14	4.02	**18.15**	1.80	20.75	0.37	0.05	(-0.43,6.65E-1)
bl-wr2-wht2.10–50.rand1	50	22.55	2.40	318.52	24.10	1.95	155.90	24.40	2.13	22.04	24.40	2.13	24.40	2.04	1.00	(-0.74,4.61E-1)
bl-wr2-wht2.10–50.rand1	125	38.10	4.37	1004.46	42.30	4.54	344.09	42.45	4.42	86.62	42.45	4.42	42.50	13.42	0.95	(-0.12,9.03E-1)
bl-wr2-wht2.10–50.rand1	200	50.85	4.89	1151.20	56.30	5.25	847.41	57.10	5.47	217.88	57.10	5.47	57.20	131.93	0.90	(-0.52,6.08E-1)
bl-wr2-wht2.10–50.rand2	50	23.85	2.78	187.17	25.25	2.70	79.43	25.45	2.60	26.37	25.45	2.60	25.45	2.15	1.00	(-0.28,7.83E-1)
bl-wr2-wht2.10–50.rand2	125	40.10	5.07	674.61	43.70	5.20	143.35	44.05	5.33	96.20	44.05	5.33	44.85	10.54	0.45	(-0.24,8.08E-1)
bl-wr2-wht2.10–50.rand2	200	54.50	4.68	1009.07	59.30	4.81	720.25	60.20	5.34	193.04	60.20	5.34	60.50	146.17	0.75	(-0.65,5.18E-1)
bl-wr2-wht2.10–50.sdeg	50	22.55	2.65	208.49	24.15	2.24	48.06	24.25	2.21	23.52	24.25	2.21	24.35	2.10	0.90	(-0.17,8.70E-1)
bl-wr2-wht2.10–50.sdeg	125	38.85	4.70	534.62	42.25	4.67	190.06	42.65	4.48	108.78	42.65	4.48	43.05	19.79	0.70	(-0.32,7.49E-1)
bl-wr2-wht2.10–50.sdeg	200	51.20	7.08	1306.13	55.70	6.54	798.97	**56.80**	7.10	230.03	56.75	7.03	57.10	161.87	0.75	(-0.57,5.73E-1)
graph3	16	15.70	0.56	0.96	15.70	0.56	0.46	0.50	0.87	2.94	15.70	0.56	15.80	0.06	0.90	(0.00,1.00E+0)
graph3	41	32.00	2.30	25.23	31.80	1.99	27.95	3.05	5.84	27.43	**33.65**	2.55	36.75	7.32	0.05	(-2.85,6.32E-3)
graph3	65	37.60	2.58	49.27	40.30	2.57	57.90	19.85	19.82	59.62	**43.35**	2.67	46.30	33.69	0.00	(-3.06,3.53E-3)
graph4	43	42.05	1.02	95.74	41.45	1.28	168.87	8.75	15.53	91.49	**42.30**	1.00	42.65	1.88	0.70	(-2.51,1.52E-2)
graph4	108	64.10	3.06	697.46	68.15	2.73	730.44	1.65	1.19	736.24	**76.10**	3.35	80.75	308.23	0.00	(-9.72,3.25E-13)
graph4	173	73.95	3.54	974.35	85.10	3.53	1111.98	1.85	1.53	1118.79	**94.85**	3.61	105.80	940.40	0.00	(-8.24,5.75E-11)
mesh15x15	22	21.40	0.66	13.39	21.15	1.42	10.62	4.35	7.45	14.87	**21.95**	0.22	22.00	1.06	0.95	(-2.28,2.96E-2)
mesh15x15	56	31.45	1.77	54.52	32.80	4.00	56.77	17.30	17.28	59.61	**37.45**	2.09	40.25	26.58	0.00	(-4.22,1.29E-4)
mesh15x15	90	36.30	2.54	60.05	43.15	4.84	112.52	24.85	21.45	114.68	**46.90**	2.51	53.20	84.22	0.00	(-1.19,2.38E-1)
mesh25x25	62	45.50	2.91	398.37	43.96	2.75	679.55	1.05	0.92	402.32	**50.60**	2.63	57.55	125.20	0.00	(-8.09,9.13E-11)
mesh25x25	156	59.95	2.82	1639.32	69.25	3.78	1845.70	2.50	1.28	1854.58	**80.95**	4.22	91.65	917.19	0.00	(-10.57,1.61E-14)
mesh25x25	250	67.45	3.14	2807.36	87.55	3.77	3165.17	3.40	1.28	3178.79	**101.00**	3.77	116.20	2534.67	0.00	(-11.21,1.77E-15)
Average Gap (%)		**13.86**			**8.63**			**40.64**			**4.47**					

Note that since LaPSO produces the tightest relaxed bounds, all of the gaps are computed using LaPSO's Upper Bounds. This figure plots the optimality gaps of each method in decreasing order. The full set of results is shown in Table 5. With regards to the poor results produced by CPLEX, particularly for the larger graphs tested such as mesh25x25, this can be attributed to the difficulty of the problem and limited CPU time given. For these problems, the MIP formulation can contain a significant number of variables and constraints, $O(|K||E|)$ and $O(|K||N|)$ respectively, where K is the number of commodities, E is the number of edges and N is the number of vertices. For these large instances, with only limited CPU runtime available, in some cases simply solving the LP relaxation takes a significant proportion of the available CPU time. In other cases, producing a good quality integer solution is also difficult, given that the heuristics used are generic MIP repair type heuristics with no knowledge of the problem structure.

4.7. Set 1 - Non comparable methods

Comparing LaPSO with GA (Hsu & Cho, 2015) and ILP + EA (Martín et al., 2020) is not able to be carried out fairly due to the differences in experimental set ups. The GA algorithm as reported in (Hsu & Cho, 2015) is run 30 times on one instance from each commodity-graph pair. This experiment design only tests 27 out of the 540 instances which are available. We were unable to discover which instances were tested, and therefore an accurate comparison is not available. For ILP + EA, there exist two commodity-graph pairings for which the solutions reported exceed the Upper Bounds calculated by both LaPSO and CPLEX. Without the exact solutions provided by the authors, we are unable to verify the exact testing methodology and criteria presented in (Martín et al., 2020). We have included a comparison with these two methods in Table 1 in the Appendix.

4.8. Set 2 - Comparable methods

LaPSO is able to achieve state-of-the-art averages on 10 out of the 54 instances tested, and equalling or bettering the current state-of-the-art average on 50 out of the 54 instances tested. Unfortunately the authors in (Pham et al., 2012) do not give standard deviations and therefore statistical tests could not be carried out. We have included the standard deviations produced by LaPSO in Table 6 for future researchers. The average GAP percentage for LaPSO is 4.00% whilst ACO, LS-SGA, LS-R and CPLEX have average gaps of 12.31%, 9.42%, 7.35% and 23.14% respectively. Again, in a similar manner as described in Section 4.6, for the larger and more difficult instances, CPLEX struggles to find either a LP solution or a good primal solution within the limited CPU time. LaPSO is able to prove optimality for 64.81% of the 1080 instances tested. A gap ranking comparison is shown in Fig. 9. This figure plots the optimality gaps of each method in decreasing order. It shows that LaPSO has the fewest instances (just 19 out of 54) that were not solved to optimality. The full set of results is shown in Table 6.

4.9. Set 2 - Non comparable methods

We are unable to provide a comparable analysis to the results given for the MP and MP-rein methods presented in (Altarelli et al., 2015), as well as the ILP + EA method presented in (Martín et al., 2020). The results given for both MP and MP-rein methods do not contain run-times for the instances tested, a significant factor

Table 6
Comparison results of the three methods ACO, LS - SGA, LS - R, CPLEX and LaPSO for Dataset 2. The first column graph describes the graph structure used. The CPU time limit for all runs was 30 minutes as described in Pham et al. (2012). $\bar{q}$ represents the average objective value over the 20 runs completed for each of the 54 graph and commodity pairings. std represent the standard deviation of the objective values, whilst $\bar{t}$ represents the average CPU runtime taken to find the **best** solution within each instance. The best $\bar{q}$ are bolded to show the best average objective value between the ACO, LS - SGA, LS - R, CPLEX and LaPSO methods. The provably optimal column shows what proportion of instances were provably optimal when using the Upper Bounds found during the LaPSO algorithm.

Graph	ACO		LS - SGA		LS - R		CPLEX			LaPSO				
	$\bar{q}$	$\bar{t}$	$\bar{q}$	$\bar{t}$	$\bar{q}$	$\bar{t}$	$\bar{q}$	std	$\bar{t}$	$\bar{q}$	std	UB	$\bar{t}$	Optimal
bl-wr2-wht2.10–50.rand.rpairs.10.1	14.80	131.60	15.60	410.76	16.00	194.71	16.00	0.00	12.63	16.00	0.00	16.00	1.12	1.00
bl-wr2-wht2.10–50.rand.rpairs.10.2	25.25	95.41	25.90	434.43	26.00	151.09	26.00	0.00	20.18	26.00	0.00	26.00	1.67	1.00
bl-wr2-wht2.10–50.rand.rpairs.25.1	31.85	165.22	31.40	564.99	32.00	263.71	32.00	0.00	82.60	32.00	0.00	32.00	5.54	1.00
bl-wr2-wht2.10–50.rand.rpairs.25.2	34.75	97.33	34.40	544.02	34.95	303.26	35.00	0.00	82.44	35.00	0.00	35.00	5.58	1.00
bl-wr2-wht2.10–50.rand.rpairs.40.1	37.85	219.56	37.60	322.96	37.90	230.29	38.00	0.00	171.10	38.00	0.00	38.00	14.41	1.00
bl-wr2-wht2.10–50.rand.rpairs.40.2	36.95	185.14	36.05	422.18	36.95	293.27	37.00	0.00	131.55	37.00	0.00	37.00	14.60	1.00
bl-wr2-wht2.10–50.sdeg.rpairs.10.1	15.95	89.24	16.25	529.03	17.00	430.99	17.00	0.00	20.07	17.00	0.00	17.00	1.64	1.00
bl-wr2-wht2.10–50.sdeg.rpairs.10.2	19.20	401.19	19.65	522.22	20.00	448.59	20.00	0.00	20.90	20.00	0.00	20.00	1.55	1.00
bl-wr2-wht2.10–50.sdeg.rpairs.25.1	35.80	67.08	35.45	536.40	36.00	423.68	36.00	0.00	91.43	36.00	0.00	36.00	4.91	1.00
bl-wr2-wht2.10–50.sdeg.rpairs.25.2	32.95	365.09	32.90	880.04	33.90	516.21	34.00	0.00	89.28	34.00	0.00	34.00	4.90	1.00
bl-wr2-wht2.10–50.sdeg.rpairs.40.1	33.65	169.90	33.10	472.56	34.00	557.57	34.00	0.00	145.06	34.00	0.00	34.00	14.01	1.00
bl-wr2-wht2.10–50.sdeg.rpairs.40.2	36.50	133.00	35.70	936.57	37.00	583.99	37.00	0.00	146.29	37.00	0.00	37.00	13.72	1.00
mesh15x15.rpairs.10.1	19.65	457.46	21.75	644.11	21.55	360.53	22.00	0.00	62.25	22.00	0.00	22.00	4.02	1.00
mesh15x15.rpairs.10.2	17.50	479.89	19.40	515.65	19.45	568.74	20.00	0.00	135.34	20.00	0.00	22.00	2.32	0.00
mesh15x15.rpairs.25.1	27.70	470.98	29.80	335.51	32.00	887.93	35.00	0.00	1804.52	35.00	0.00	37.00	609.65	0.00
mesh15x15.rpairs.25.2	29.20	1010.52	31.70	480.98	33.05	592.96	35.00	0.00	378.07	35.00	0.00	37.15	52.84	0.00
mesh15x15.rpairs.40.1	35.30	871.22	35.80	763.25	38.80	960.97	**42.00**	0.00	1805.63	41.00	0.00	46.00	74.47	0.00
mesh15x15.rpairs.40.2	34.00	750.55	34.60	649.26	37.60	910.63	**42.00**	0.00	1806.33	41.00	0.00	45.00	1011.78	0.00
mesh25x25.rpairs.10.1	32.85	996.96	39.15	864.60	41.00	946.47	0.00	0.00	1804.58	**48.00**	0.00	55.00	61.13	0.00
mesh25x25.rpairs.10.2	30.10	944.36	35.70	875.12	37.90	945.08	0.90	0.30	1804.92	**47.00**	0.00	53.00	142.62	0.00
mesh25x25.rpairs.25.1	45.00	1104.82	51.95	1053.57	55.55	1111.80	0.00	0.00	1805.93	**66.00**	0.00	75.00	1468.14	0.00
mesh25x25.rpairs.25.2	45.60	1015.84	51.35	673.59	54.70	1042.11	0.00	0.00	1806.48	**66.00**	0.00	73.35	1517.50	0.00
mesh25x25.rpairs.40.1	57.70	797.14	65.30	950.87	69.30	1520.61	0.00	0.00	1811.66	**83.00**	0.00	105.00	1806.39	0.00
mesh25x25.rpairs.40.2	57.75	939.82	65.05	1409.13	68.85	1040.24	0.00	0.00	1812.61	**82.30**	0.46	106.55	1777.54	0.00
planar-n100.ins1.rpairs.10.1	10.00	0.12	10.00	1.14	10.00	1.07	10.00	0.00	2.16	10.00	0.00	10.00	0.03	1.00
planar-n100.ins1.rpairs.25.1	25.00	20.22	25.00	7.05	25.00	5.33	25.00	0.00	19.98	25.00	0.00	25.00	0.08	1.00
planar-n100.ins1.rpairs.40.1	34.00	680.72	35.30	813.56	36.00	698.88	37.00	0.00	166.30	37.00	0.00	40.00	5.40	0.00
planar-n200.ins1.rpairs.10.1	20.00	13.46	20.00	4.06	20.00	5.23	20.00	0.00	4.98	20.00	0.00	20.00	0.05	1.00
planar-n200.ins1.rpairs.25.1	41.80	889.07	44.85	988.81	45.95	853.18	50.00	0.00	908.02	50.00	0.00	50.00	14.47	1.00
planar-n200.ins1.rpairs.40.1	49.35	790.65	53.35	1033.97	55.70	901.74	62.00	0.00	1803.41	62.00	0.00	65.00	851.21	0.00
planar-n50.ins1.rpairs.10.1	5.00	0.03	5.00	0.86	5.00	0.80	5.00	0.00	2.08	5.00	0.00	5.00	0.01	1.00
planar-n50.ins1.rpairs.25.1	12.00	0.16	12.00	0.96	12.00	0.97	12.00	0.00	2.38	12.00	0.00	12.00	0.01	1.00
planar-n50.ins1.rpairs.40.1	20.00	36.38	20.00	25.12	19.90	31.18	20.00	0.00	7.75	20.00	0.00	20.00	0.02	1.00
planar-n500.ins1.rpairs.10.1	44.95	1100.41	49.95	484.84	50.00	309.24	46.07	12.30	1778.53	50.00	0.00	50.00	0.96	1.00
planar-n500.ins1.rpairs.25.1	60.95	954.35	73.85	1345.74	78.20	1044.03	0.00	0.00	1805.59	**94.00**	0.00	103.00	653.41	0.00
planar-n500.ins1.rpairs.40.1	82.85	1235.13	93.95	1366.27	100.15	1455.43	3.00	0.00	1807.94	**121.00**	0.00	137.00	1662.02	0.00
steinb10.rpairs.10.1	7.00	0.02	7.00	1.35	7.00	1.16	7.00	0.00	1.83	7.00	0.00	7.00	0.00	1.00
steinb10.rpairs.25.1	17.85	96.48	18.00	13.42	18.00	5.20	18.00	0.00	4.24	18.00	0.00	18.00	0.03	1.00
steinb10.rpairs.40.1	24.35	242.04	26.25	682.84	27.30	505.22	**28.00**	0.00	1803.26	27.00	0.00	30.00	2.36	0.00
steinb16.rpairs.10.1	10.00	0.25	10.00	1.46	10.00	1.52	10.00	0.00	2.03	10.00	0.00	10.00	0.02	1.00
steinb16.rpairs.25.1	24.35	364.99	25.00	93.53	25.00	8.86	25.00	0.00	8.78	25.00	0.00	25.00	0.16	1.00
steinb16.rpairs.40.1	32.45	658.25	34.10	747.34	35.95	646.19	**36.00**	0.00	1803.21	35.00	0.00	40.00	3.55	0.00
steinb4.rpairs.10.1	5.00	0.01	5.00	1.12	5.00	1.09	5.00	0.00	1.65	5.00	0.00	5.00	0.00	1.00
steinb4.rpairs.25.1	12.00	0.44	12.00	1.22	12.00	1.40	12.00	0.00	1.82	12.00	0.00	12.00	0.01	1.00
steinb4.rpairs.40.1	20.00	51.11	20.00	5.45	19.90	2.80	20.00	0.00	3.64	20.00	0.00	20.00	0.01	1.00
steinc11.rpairs.10.1	50.00	23.59	50.00	42.19	50.00	37.64	50.00	0.00	21.92	50.00	0.00	50.00	0.58	1.00
steinc11.rpairs.25.1	123.30	521.80	125.00	128.49	125.00	262.38	2.70	0.90	1805.40	125.00	0.00	125.00	4.55	1.00
steinc11.rpairs.40.1	194.25	494.64	200.00	395.54	200.00	473.81	2.70	0.90	1810.58	200.00	0.00	200.00	16.07	1.00
steinc16.rpairs.10.1	50.00	6.89	50.00	55.12	50.00	46.01	50.00	0.00	76.54	50.00	0.00	50.00	1.26	1.00
steinc16.rpairs.25.1	125.00	17.13	125.00	194.36	125.00	113.83	125.00	0.00	385.62	125.00	0.00	125.00	10.50	1.00
steinc16.rpairs.40.1	200.00	45.32	200.00	366.69	200.00	183.32	200.00	0.00	1820.29	200.00	0.00	200.00	32.18	1.00
steinc6.rpairs.10.1	49.10	572.75	50.00	184.44	50.00	240.75	50.00	0.00	33.12	50.00	0.00	50.00	0.43	1.00
steinc6.rpairs.25.1	89.90	728.76	92.20	734.88	104.95	1370.88	1.00	0.00	1805.94	**115.00**	0.00	125.00	442.79	0.00
steinc6.rpairs.40.1	109.80	924.10	112.05	971.40	121.40	1372.37	3.00	0.00	1806.88	**136.00**	0.00	177.00	1462.60	0.00
Average Gap (%)	**12.31**		**9.42**		**7.35**		**24.97**			**4.00**				

affecting the quality of solutions. We therefore do not consider these results to be a fair comparison for state-of-the-art purposes. Regarding the ILP + EA method, there exist similar problems to those described for Set 1. In (Martín et al., 2020) the results given show 21 out of the 54 instances tested achieving primal solution values greater than the Upper Bounds of both LaPSO and CPLEX. Without having access to the authors solutions in (Martín et al., 2020), we are unable to comment on these differences. We have included both MP, MP-rein and ILP + EA methods in Table 2 in the Appendix for completeness.

4.10. Set 3 - Newly created larger graphs

For the newly created instances in Dataset 3, we can see more clearly the effectiveness of LaPSO when compared to CPLEX as problem sizes are increased, as shown in Table 7. During testing, we discovered that synthetically creating new problem instances which are difficult to solve is quite a challenging task in and of itself. Despite the significant increase in graph sizes, as seen in the "easy" labelled instances, LaPSO is able to find almost optimal solutions for most of the instances tested, despite only being able

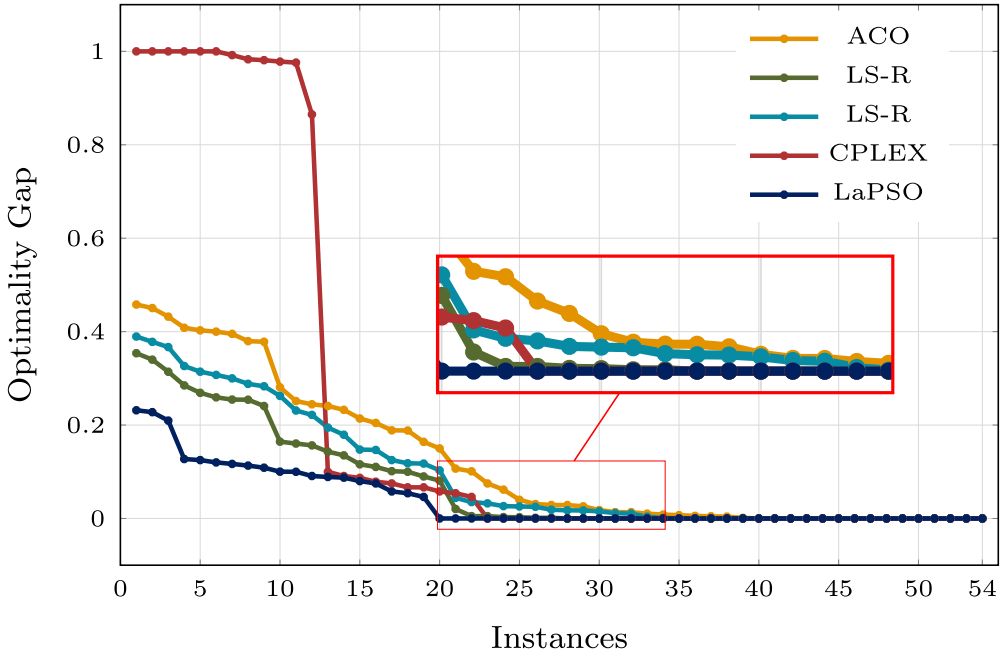


Fig. 9. Optimality Gap Comparisons. Optimality gaps calculated using the average heuristic Lower Bounds and LaPSO's Upper Bounds from 20 runs for each of the 54 graph-commodity pairs in Dataset 2. Optimality gaps are plotted in descending order for each method. The Instances axis does not refer to a specific instance, instead it is the index in the ordered sequence.

Table 7
Comparison results of the two methods CPLEX and LaPSO for Dataset 3. The first column graph describes the graph structure used, with the ODs column describing the number of commodities to be transported. $\bar{q}$ represents the average objective value over the 20 different problem instances for each graph and commodity pairing. std represents the standard deviation of the objective values and $\bar{t}$ represents the average CPU runtime for each instance solved using LaPSO. Whilst a runtime limit of 3600 CPU seconds was given for both CPLEX and LaPSO, in some instances LaPSO exceeds this runtime as it completes its current iteration. We therefore include $\bar{t}$ as it is a more fair CPU time for future researchers to benchmark against. The best $\bar{q}$ are bolded to show the best average objective value between the CPLEX and LaPSO methods. The Optimal column shows what proportion of instances were provably optimal when using the best Upper Bounds found by CPLEX or the LaPSO algorithm.

Graph	ODs	CPLEX			LaPSO				
		$\bar{q}$	std	UB	$\bar{q}$	std	UB	$\bar{t}$	Optimal
easy_2_1	400	0.00	0.00	398.00	**398.00**	0.00	399.00	3627.97	1.00
easy_2_2	400	0.00	0.00	400.00	**400.00**	0.00	400.00	295.39	1.00
easy_3_1	600	0.00	0.00	598.00	**598.00**	0.00	600.00	3829.58	1.00
easy_3_2	600	0.00	0.00	600.00	**600.00**	0.00	600.00	1034.52	1.00
easy_4_1	800	0.00	0.00	797.00	**795.00**	0.00	800.00	3808.77	0.00
easy_4_2	800	0.00	0.00	799.00	**797.15**	0.36	800.00	3837.09	0.00
easy_5_1	1000	0.00	0.00	999.00	**996.00**	0.00	1000.00	4175.43	0.00
easy_5_2	1000	0.00	0.00	999.00	**996.00**	0.00	1000.00	4185.91	0.00
Average Gap (%)		100			**0.13**				
hard-small	475	0.00	0.00	389.00	**292.00**	0.00	293.00	3609.75	0.00
hard-med	725	0.00	0.00	718.00	**501.30**	0.46	667.00	3661.13	0.00
hard-large	975	0.00	0.00	954.00	**711.00**	0.00	975.00	3896.16	0.00
Average Gap (%)		100			**16.89**				

to run a very limited number of iterations. This suggests to us that these instances are too easy for a heuristic based path finding technique, perhaps indicating that the number of paths available for commodities to be routed are too numerous. For a Mixed Integer Programming (MIP) solver however, the increase in problem size simply becomes too difficult to provide any meaningful bounds or heuristic solutions. Within the given runtime (1 h and), the only feasible solutions produced by CPLEX are the incumbent solutions with values of 0. This can be attributed to the scale of these problem sizes, as even solving the root node relaxation to optimality is often not even possible within the available runtime. Considering the number of variables is $O(|K||E|)$, where K is the number of commodities and E is the number of edges, for the largest "easy" instances, the number of variables is close to ten million. In addition to the large number of variables, the MEDP problem also contains a significant number of constraints, $O(|K||V|)$, where V is the number of vertices, and therefore solving even the relaxed problem is difficult, using either a primal or dual simplex approach.

For the "hard" graphs, we can start to see what problem sizes result in a degradation of the quality of solutions produced by LaPSO. Whilst the optimality gap for the hard-small instance is 0.34%, for the hard-med and hard-large instances, the optimality gaps are 24.84% and 27.08% respectively. Looking at Fig. 10 it is clear that there simply isn't enough CPU time available to solve difficult instances of this size, with only a limited number of iterations of the LaPSO algorithm being able to be completed. Despite the problem sizes of Dataset 3 being larger than the problem

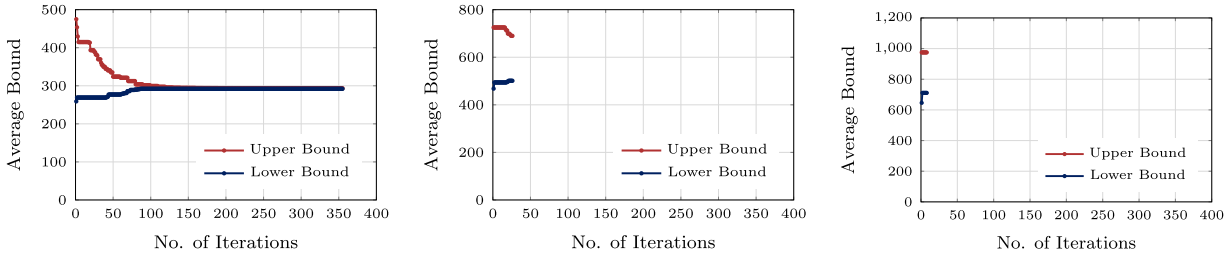


Fig. 10. Bound comparisons for the hard instances in the newly created Dataset. The bound values given are averages of the best upper and lower bounds for the 20 different runs of LaPSO carried out.

sizes in Dataset 2 in only a linear fashion, coupled with the doubling of the CPU runtime limit (from 1800s to 3600s), because the shortest path algorithm used to solve the subproblems in LaPSO is non linear itself ($O(|E \log V|)$, this has resulted in significantly fewer iterations being able to be performed within the given runtime. For both "easy" and "hard" instances, we can still see the effectiveness of the LaPSO method being a "quasi-exact" method, in the sense that it is able to produce both good primal solutions and bounds for the MEDP problem, thereby providing an optimality gap which heuristic based methods are unable to. For "easy" problems, LaPSO has shown to be effective for problem sizes up to 2500 nodes and 10,000 edges. For more difficult problems, LaPSO's effectiveness has been demonstrated for problem sizes containing between 500–1000 nodes and 1500–3000 edges, within a one hour runtime limit.

5. Conclusions

This paper presented an adaptation of the LaPSO algorithm in the context of the Maximum Edge Disjoint Path (MEDP) problem. LaPSO is a hybrid metaheuristic combining Lagrangian Relaxation with Particle Swarm Optimisation in a way that is generically applicable but has not yet been widely tested. A novel repair heuristic was presented, which we have labelled LVP (Largest Violation Perturbation). We have shown that this new repair method significantly outperforms other alternatives when embedded in the LaPSO framework. We have also provided analysis regarding the effect that the perturbations have on the LaPSO algorithm. Our computational results show that the LaPSO algorithm outperforms current state-of-the-art methods, producing better primal solutions for 32.1% of the instances tested, and similar solution qualities for the remainder. Unlike other heuristic based techniques, the LaPSO algorithm is also able to produce relaxed bounds that can provide optimality guarantees. This allows the LaPSO algorithm to act as a "quasi-exact" method, producing provably optimal solutions for most of the problem instances tested. This benefit is highlighted when testing the larger, non-trivial instances, where a more traditional "exact" method such as a Mixed Integer Programming (MIP) solver is unable to produce any meaningful bounds when given limited CPU time. We have also created a new Dataset for benchmarking purposes, containing instances significantly larger than previous benchmarks. For these newly created instances, the effectiveness of LaPSO being able to provide both high quality primal and dual solutions is even more pronounced, when compared to the results generated by a MIP solver. It is interesting that our heuristic method not only outperforms other meta-heuristic algorithms but also outperforms linear programming based exact solvers (CPLEX and the Volume Algorithm) in finding good relaxed bounds. Hence, while the current work has focused mainly on the ability to produce high quality heuristic solutions, future work may look at producing tighter bounds so that more problems can be solved to provable optimality. The strong performance of our LaPSO method for this problem also suggests that the same framework may yield high performing results for other combinatorial optimisation problems which are amenable to a Lagrangian relaxation based approach.

Acknowledgements

This research was supported by an ARC (Australian Research Council) Discovery Grant (DP190101271).

Supplementary material

Supplementary material associated with this article can be found, in the online version, at doi:10.1016/j.ejor.2021.01.009.

References

Altarelli, F., Braunstein, A., Dall'Asta, L., De Bacco, C., & Franz, S. (2015). The edge-disjoint path problem on random graphs by message-passing. *PloS One, 10*(12), e0145222.
Awerbuch, B., Gawlick, R., Leighton, F. T., & Rabani, Y. (1994). On-line Admission Control and Circuit Routing for High Performance Computing and Communication. In *35th annual symposium on foundations of computer science, santa fe, new mexico, usa, 20–22 november 1994* (pp. 412–423). IEEE Computer Society.
Barahona, F., & Anbil, R. (2000). The volume algorithm: Producing primal solutions with a subgradient method. *Mathematical Programming, 87*(3), 385–399.
Blesa, M. J., & Blum, C. (2007). Finding edge-disjoint paths in networks by means of artificial ant colonies. *Journal of Mathematical Modelling and Algorithms, 6*(3), 361–391.
Chan, W.-T., Chin, F. Y. L., & Ting, H.-F. (2003). Escaping a grid by edge-disjoint paths. *Algorithmica, 36*(4), 343–359.
Dijkstra, E. W. (1959). A note on two problems in connexion with graphs. *Numerische mathematik, 1*(1), 269–271.
Eberhart, R., & Kennedy, J. (1995). A new optimizer using particle swarm theory. In *Proceedings of the sixth international symposium on micro machine and human science* (pp. 39–43).
Ernst, A. T. (2010). A hybrid Lagrangian Particle Swarm Optimization Algorithm for the degree-constrained minimum spanning tree problem. In *Proceedings of the IEEE congress on evolutionary computation* (pp. 1–8).
Ernst, A. T., & Singh, G. (2012). Lagrangian Particle Swarm Optimization for a resource constrained machine scheduling problem. In *Proceedings of the IEEE congress on evolutionary computation* (pp. 1–8).
Fisher, M. L. (1981). The Lagrangian relaxation method for solving integer programming problems. *Management Science, 27*(1), 1–18.
Gerez, S. H. (1998). *Algorithms for VLSI design automation*. United States: Wiley.
Haghighi, S. (2017). Pyrgg: Python random graph generator. *Journal of Open Source Software, 2*(17), 331.
Hiriart-Urruty, J.-B., & Lemaréchal, C. (1993). Convex analysis and minimization algorithms. I, volume 305 of Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences].
Hsu, C.-C., & Cho, H.-J. (2015). A genetic algorithm for the maximum edge-disjoint paths problem. *Neurocomputing, 148*, 17–22.
Jain, S., & Das, S. R. (2005). Exploiting path diversity in the link layer in wireless ad hoc networks. In *Proceedings of the Sixth IEEE international symposium on a world of wireless mobile and multimedia networks* (pp. 22–30).
Karp, R. M. (1972). Reducibility among combinatorial problems. In *Complexity of computer computations* (pp. 85–103). Springer.
Kleinberg (1996). *Approximation Algorithms for Disjoint Paths Problems*. USA Ph.D. thesis..
Kolliopoulos, S. G., & Stein, C. (2004). Approximating disjoint-path problems using packing integer programs. *Mathematical Programming, 99*(1), 63–87.

Li, X., & Cuthbert, L. (2004). Node-disjointness-based multipath routing for mobile ad hoc networks. In *Proceedings of the 1st ACM international workshop on performance evaluation of wireless ad hoc, sensor, and ubiquitous networks* (pp. 23–29). ACM.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., & Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives, 3*, 43–58.

Martín, B., Sánchez, Á., Beltran-Royo, C., & Duarte, A. (2020). Solving the edge-disjoint paths problem using a two-stage method. *International Transactions in Operational Research, 27*(1), 435–457.

Mézard, M., & Parisi, G. (2003). The cavity method at zero temperature. *Journal of Statistical Physics, 111*(1–2), 1–34.

Pham, Q. D., Deville, Y., & Van Hentenryck, P. (2012). LS (Graph): a constraint-based local search for constraint optimization on trees and paths. *Constraints, 17*(4), 357–408.

Poli, R., Kennedy, J., & Blackwell, T. (2007). Particle swarm optimization. *Swarm Intelligence, 1*(1), 33–57.

Raghavan, P., & Upfal, E. (1994). Efficient Routing in All-Optical Networks. In *Proceedings of the twenty-sixth annual acm symposium on theory of computing*. In *STOC '94* (pp. 134–143). New York, NY, USA: Association for Computing Machinery.

Shi, Y., & Eberhart, R. (1998). A modified particle swarm optimizer. In *Proceedings of the IEEE international conference on evolutionary computation proceedings. ieee world congress on computational intelligence (cat. no.98th8360)* (pp. 69–73).

Sumpter, Z., Burson, L., Tang, B., & Chen, X. (2013). Maximizing number of satisfiable routing requests in static ad hoc networks. In *Proceedings of the IEEE global communications conference (globecom)* (pp. 19–24). IEEE.

Wedelin, D. (1995). An algorithm for large scale 0-1 integer programming with application to airline crew scheduling. *Annals of Operations Research, 57*(1), 283–301.

Zhao, X., & Luh, P. B. (2002). New bundle methods for solving Lagrangian relaxation dual problems. *Journal of Optimization Theory and Applications, 113*(2), 373–397.