COMPUTER TECHNOLOGY INSTITUTE

TECHNICAL REPORT No. TR 2004/03/01

NON-ADDITIVE SHORTEST PATHS

George Tsaggouris Christos D. Zaroliagis

March 2004

Non-Additive Shortest Paths*

George Tsaggouris† Christos Zaroliagis†

March 22, 2004

Abstract

The non-additive shortest path (NASP) problem asks for finding an optimal path that minimizes a certain multi-attribute non-linear cost function. In this paper, we consider the case of a non-linear convex and non-decreasing function on two attributes. We present an efficient polynomial algorithm for solving a Lagrangian relaxation of NASP. We also present an exact algorithm for solving the problem that is based on new heuristics we introduce here, and conduct a comparative experimental study with synthetic and real-world data that demonstrates the quality of our approach.

1 Introduction

The shortest path problem is one of the most fundamental network optimization problems encompassing a vast number of applications. Given a digraph $G = (V, E)$ with a cost function on its edges, the shortest path problem asks for finding a path between two nodes s and t of total minimum cost. The problem has been heavily studied under the *additivity assumption* stating that the cost of a path is the sum of the costs of its edges. In certain applications, however, the cost function may not be additive along paths and may not depend on a single attribute. This gives rise to the so-called *non-additive shortest path* (NASP) problem, which asks for finding an optimal path that minimizes a certain multi-attribute *non-linear cost function*.

In this work, we are interested in the version of NASP in which every edge (and hence every path) is associated with two attributes, say cost and resource, and in which the non-linear objective function is convex and non-decreasing. This version of NASP has numerous applications in bicriteria network optimization problems, and especially in transportation networks; for example, it is the core subproblem in finding traffic equilibria [4, 12]. In such applications, a utility cost function for a path p is defined that typically translates travel time (and travel cost if necessary) to a common utility cost measure (e.g., money). Experience shows that users of traffic networks value time non-linearly [8]: small amounts of time have relatively low value, while large amounts of time are very valuable; consider, e.g., the transportation of a sensitive in time commodity (like milk), or the transportation of a hazardous material, or the response time to an emergency incident by an emergency service system. Also, the vast majority of toll road or transit systems have a non-additive (non-linear) toll/fare structure [4]. Consequently, the most interesting theoretical models for traffic equilibria [4, 12] involve minimizing a monotonic non-linear utility function of two attributes, travel cost and travel time, that allows a decision maker to find the optimal path both w.r.t. travel cost and time.

*This work was partially supported by the IST Programme (6th FP) of EC under contract No. IST-2002-001907 (integrated project DELIS).

†Computer Technology Institute, P.O. Box 1122, 26110 Patras, Greece, and Dept of Computer Engineering and Informatics, University of Patras, 26500 Patras, Greece. Emails: {tsaggour,zaro}@ceid.upatras.gr

NASP is related to multiobjective optimization, and actually appears to be a core problem in this area; e.g., in multi-agent competitive network models [5]. The goal in multiobjective optimization problems is to compute a set of solutions (paths in the case of NASP) optimizing several cost functions (also called criteria). These problems are typically solved by producing the so-called *Pareto curve* [11], which is the set of all undominated Pareto-optimal solutions or "trade-offs" (the set of feasible solutions where the attribute-vector of one solution is not dominated by the attribute-vector of another solution). Multiobjective optimization problems are usually NP-hard. The difficulty in solving them stems from the fact that the Pareto curve is typically exponential in size, and this applies to NASP itself. Note that even if a decision maker is armed with the entire set of all undominated Pareto-optimal solutions, s/he is still left with the problem of which is the "best" solution for the application at hand. Consequently, three natural approaches to solve multiobjective optimization problems are: (i) optimize one objective while bounding the rest (cf. e.g., [2, 6, 9]); (ii) find approximation schemes that generate a polynomial number of solutions which are within a small factor in all objectives [11]; (iii) proceed in a normative way and choose the "best" solution by introducing a utility function on the attributes involved (for example, a function that values time for money as in the aforementioned transportation applications).

In this paper, we concentrate on devising an exact algorithm for solving NASP as well as on computing a Lagrangian relaxation of the the integer non-linear programming (INLP) formulation of the problem, which is the first step in finding the optimal solution. Solving the Lagrangian relaxation of NASP is equivalent to computing the "best" path among those that lie on the convex hull of the Pareto curve, called extreme paths. Note that these paths may also be exponentially many and that the best extreme path is not necessarily the optimal [10]. To find the best extreme path, we adopt the so-called hull approach, which is a cutting plane method that has been used in various forms and by various researchers since 1966 [3]. Its first explicit use in bicriteria network optimization was made in [6] for solving the Resource Constrained Shortest Path (RCSP)[1] problem. However, only recently a considerable refinement of the method has been made as well as its worst-case complexity was carefully analyzed [9]. Variations of the hull approach for solving relaxations of NASP, when the non-linear objective function is convex and non-decreasing, have previously appeared in [7, 10], but with very crude time analyses giving bounds proportional to the number of the extreme Pareto paths. Regarding the exact solution to NASP, we adopt and extend the classical 3-phase approach introduced in [6] for solving RCSP, used later in [2], and considerably refined in [9]: (a) compute lower and upper bounds to the optimal solution, using the solution of the relaxed problem; (b) prune the graph by deleting nodes and edges that cannot belong to an optimal path; (c) close the gap between upper and lower bound. The pruning phase was not considered in [6, 7]. Other approaches for solving NASP, either involved heuristics based on path flow formulations and repeatedly solving LPs [4, 5], or were based on repeated applications of RCSP [12] (one for each possible value of the resource of a path).

In this work, we provide new theoretical and experimental results for solving NASP. More specifically, our contributions are threefold: (i) We extend the hull algorithm in [9] and provide an efficient polynomial time algorithm for solving the Lagrangian relaxation of NASP. Note that the extension of a method that solves the relaxation of an integer linear program (RCSP) to a method that solves the relaxation of an integer non-linear program (NASP) is not straightforward. For integral costs and resources in $[0, C]$ and $[0, R]$, respectively, our algorithm runs in $O(\log(nRC)(m + n \log n))$ time, where n and m are the number of nodes and edges of the graph. To the best of our knowledge, this is the first efficient algorithm for solving a relaxation of NASP. Our algorithm provides both upper and lower bounds, while the one in [10] provides only the former. (ii) We give an exact algorithm to find the optimum of NASP based on the aforementioned 3-phase approach.

[1] RCSP asks for finding a minimum cost path such that its resource does not exceed a specific bound. Both cost and resource are *additive* along paths. RCSP is modelled as an ILP.

Except for the relaxation method, our algorithm differentiates from those in [6, 7, 9, 10] also in the following: (a) we provide two new heuristics, the *gradient* and the *on-line node reductions* heuristics, that can be plugged in the extended hull algorithm and considerably speed up the first phase as our experimental study shows. (b) We give a method for gap closing that is not based on k-shortest paths, as those in [6, 7], but constitutes an extension and improvement of the method for gap closing given in [9] for the RCSP problem. Again, our experiments show that our method is faster than those in [6, 7, 9]. (iii) We introduce a generalization of both NASP and RCSP, in which one wishes to minimize a non-linear, convex and non-decreasing function of two attributes such that both attributes do not exceed some specific bounds, and show that the above results hold for the generalized problem, too.

2 Preliminaries and Problem Formulation

In the following, let $G = (V, E)$ be a digraph with n nodes and m edges, and let $c : E \rightarrow I\!R_0^+$ be a function associating non-negative costs to the edges of G. The cost of an edge $e \in E$ will be denoted by $c(e)$. To better introduce the definition of NASP, we start with an Integer Linear Programming (ILP) formulation of the well-known additive shortest path problem given in [9].

Let P be the set of paths from node s to node t in G. For each path $p \in P$ we introduce a 0-1 variable x_p, and we denote by c_p, or by $c(p)$, the path's cost, i.e., $c_p \equiv c(p) = \sum_{e \in p} c(e)$. Then, the ILP for the additive shortest path is as follows.

$$\min \quad \sum_{p \in P} c_p x_p \tag{1}$$

$$s.t. \quad \sum_{p \in P} x_p = 1 \tag{2}$$

$$x_p \in \{0, 1\}, \forall p \in P \tag{3}$$

Suppose now that, in addition to the cost function c, there is also a resource function $r : E \rightarrow I\!R_0^+$ associating each edge $e \in E$ with a non-negative resource $r(e)$ (representing e.g., time). We denote the resource of a path p by r_p, or by $r(p)$, and define it as $r_p \equiv r(p) = \sum_{e \in p} r(e)$. Consider now a non-decreasing and convex function $U : I\!R_0^+ \rightarrow I\!R_0^+$ with first derivative $U' : I\!R_0^+ \rightarrow I\!R_0^+$. Function U is a translation function that converts the resource of a path to cost. Now, the shortest path problem asks for finding a path $p \in P$ which minimizes the overall cost, that is:

$$\min \sum_{p \in P} c_p x_p + U(\sum_{p \in P} r_p x_p) \tag{4}$$

Substituting (1) with the new objective function (4) and keeping the same constraints (2) & (3), we get an Integer Non-Linear Program (INLP). Since the overall cost of a path is no longer additive on its edges the problem is called the *non-additive shortest path problem* (NASP). It is easy to see that the solution to NASP is a Pareto-optimal path. We would like to mention that in the full generality of the problem there is also a translation function for cost (cf. Section 5). We start from this simpler objective function, however, for two reasons: (a) it is the most commonly used in applications (see e.g., [4, 5, 12]); and (b) it better introduces our approach for the general case.

3 Lagrangian Relaxation and its Solution

We can transform the above INLP into an equivalent one by introducing a variable z and setting it equal to $z = \sum_{p \in P} r_p x_p$. Thus the INLP objective function becomes min $\sum_{p \in P} c_p x_p + U(z)$, while an additional equality constraint is introduced: $\sum_{p \in P} r_p x_p = z$. Since now $U(\cdot)$ is a non-decreasing

function we can relax this equality constraint to inequality, and get the following INLP which is equivalent to the original one.

$$\begin{aligned} \min \quad & \sum_{p\in P} c_p x_p + U(z) \\ s.t. \quad & \sum_{p\in P} r_p x_p \le z \\ & \sum_{p\in P} x_p = 1 \\ & x_p \in \{0,1\}, \forall p \in P \end{aligned} \tag{5}$$

We can relax the above problem using Lagrangian relaxation. That is, by introducing a Lagrangian multiplier $\mu \ge 0$, multiplying the two sides of (5) by μ, and turning constraint (5) in the objective function.

$$\begin{aligned} L(\mu) \;=\; \min \quad & \sum_{p\in P} (c_p + \mu r_p)x_p + U(z) - \mu z \\ s.t. \quad & \sum_{p\in P} x_p = 1 \\ & x_p \in \{0,1\}, \forall p \in P \end{aligned}$$

$L(\mu)$ is a lower bound on the optimal value of the objective function (4) for any value of the Lagrangian multiplier μ [1, Ch. 16]. A best lower bound can be found by solving the problem $\max_{\mu\ge 0} L(\mu)$ (equivalent to solving the *NLP relaxation* of NASP, obtained by relaxing the integrality constraints). Since none of the constraints in $L(\mu)$ contains variable z, the problem decomposes in two separate problems $L_p(\mu)$ and $L_z(\mu)$ such that

$$L(\mu) = L_p(\mu) + L_z(\mu),$$

$$\begin{aligned} \text{where} \qquad L_p(\mu) \;=\; \min \quad & \sum_{p\in P} (c_p + \mu r_p)x_p \\ s.t. \quad & \sum_{p\in P} x_p = 1 \\ & x_p \in \{0,1\}, \forall p \in P \\ \text{and} \qquad L_z(\mu) \;=\; \min \quad & U(z) - \mu z \end{aligned} \tag{6}$$

Observe now that for a fixed μ: (i) $L_p(\mu)$ is a standard additive shortest path problem w.r.t. the composite (non-negative) edge weights $w(\cdot) = c(\cdot) + \mu r(\cdot)$ and can be solved using Dijkstra's shortest path algorithm in $O(m + n \log n)$ time; (ii) $L_z(\mu)$ is a standard minimization problem of a (single variable) convex function. These crucial observations along with some technical lemmata will allow us to provide an efficient polynomial algorithm for solving the relaxed version (Lagrangian relaxation) of the above INLP.

3.1 The Ingredients

In the following, let $p = SP(s, t, c, r, \mu)$ denote a shortest path routine that w.l.o.g. always returns the same s-t path p that is shortest w.r.t. the edge weights $w(e) = c(e) + \mu r(e)$, $\forall e \in E$, and let $\arg\min\{f(z)\}$ denote the maximum z that minimizes $f(z)$. The next lemma relates resources and costs of shortest paths w.r.t. different μ.

Lemma 1 *Let* $p_l = SP(s,t,c,r,\mu_l)$, $p_h = SP(s,t,c,r,\mu_h)$, $z_l = \arg\min\{U(z) - \mu_l z\}$, $z_h = \arg\min\{U(z) - \mu_h z\}$, *and* $0 \leq \mu_l \leq \mu_h$. *Then, (i)* $r(p_l) \geq r(p_h)$; *(ii)* $c(p_l) \leq c(p_h)$; *(iii)* $r(p_l) - z_l \geq r(p_h) - z_h$; *and (iv)* $c(p_l) + U(z_l) \leq c(p_h) + U(z_h)$.

Proof. If $\mu_l = \mu_h$, then the lemma holds trivially. Otherwise, by definition of p_l and p_h:

$$c(p_l) + \mu_l r(p_l) \leq c(p_h) + \mu_l r(p_h) \tag{7}$$
$$c(p_h) + \mu_h r(p_h) \leq c(p_l) + \mu_h r(p_l) \tag{8}$$

(i) Adding inequalities (7) and (8) and by cancelling terms that appear on both sides, we get $\mu_l r(p_l) + \mu_h r(p_h) \leq \mu_l r(p_h) + \mu_h r(p_l) \Rightarrow (\mu_h - \mu_l) r(p_h) \leq (\mu_h - \mu_l) r(p_l) \Rightarrow r(p_h) \leq r(p_l)$.
(ii) From inequality (7) we get $c(p_l) \leq c(p_h) + \mu_l r(p_h) - \mu_l r(p_l) = c(p_h) + \mu_l (r(p_h) - r(p_l)) \leq c(p_h)$.
(iii) By the definition of z_l and z_h: $U(z_l) - \mu_l z_l \leq U(z_h) - \mu_l z_h$ and $U(z_h) - \mu_h z_h \leq U(z_l) - \mu_h z_l$. Adding theses inequalities, we get $z_l \leq z_h$.
(iv) We have shown in (iii) above that $z_l \leq z_h$. Since $U(\cdot)$ is non-decreasing, we have that $U(z_l) \leq U(z_h)$. Combining this inequality with (ii) yields $c(p_l) + U(z_l) \leq c(p_h) + U(z_h)$. ∎

Lemma 1 allows us to use any line-search method to either find an optimal solution or a lower bound to the problem. Here, we follow the framework of the hull approach and in particular of its refinement in [9] for solving the RCSP problem (hull algorithm), and extend it to NASP. The approach can be viewed as performing a kind of binary search between a feasible and an infeasible solution w.r.t. constraint (5). In each step the goal is to find a pair (p,z) that either corresponds to a new feasible solution that reduces the objective value $(c(p) + U(z))$, or corresponds to an infeasible solution that decreases the amount $(r(p) - z)$ by which constraint (5) is violated.

The next two lemmata allow the effective extension of the hull approach to NASP. The first lemma shows how such a solution can be found by suitably choosing the Lagrangian multiplier at each iteration.

Lemma 2 *Let* $p_l = SP(s,t,c,r,\mu_l)$, $z_l = \arg\min\{U(z) - \mu_l z\}$, $p_h = SP(s,t,c,r,\mu_h)$, $z_h = \arg\min\{U(z) - \mu_h z\}$, $p_m = SP(s,t,c,r,\mu_m)$, $z_m = \arg\min\{U(z) - \mu_m z\}$, $0 \leq \mu_l \leq \mu_h$, *and* $\mu_m = \frac{c(p_h) - c(p_l)}{r(p_l) - r(p_h)}$. *Then, (i)* $r(p_l) - z_l \geq r(p_m) - z_m \geq r(p_h) - z_h$, *and (ii)* $c(p_l) + U(z_l) \leq c(p_m) + U(z_m) \leq c(p_h) + U(z_h)$.

Proof. By Lemma 1, we only need to prove that $\mu_l \leq \mu_m \leq \mu_h$. From inequalities (7) and (8), we get that $(r(p_l) - r(p_h))\mu_l \leq c(p_h) - c(p_l)$ and $(r(p_l) - r(p_h))\mu_h \geq c(p_h) - c(p_l)$. Moreover, since $r(p_l) \geq r(p_h)$ and $c(p_h) \geq c(p_l)$, we have that $\mu_l \leq \frac{c(p_h) - c(p_l)}{r(p_l) - r(p_h)} = \mu_m \leq \mu_h$. ∎

The previous lemma can be usefully applied when we have an efficient way to test for feasibility. Compared to RCSP, where this test was trivial (just checking whether the resource of a path is smaller than a given bound), checking whether (p,z) is feasible reduces in finding the inverse of $U'(z)$ (as objective (6) dictates) and this may not be trivial. The next lemma allows us not only to efficiently test for feasibility, but also to avoid finding the inverse of $U'(z)$.

Lemma 3 *Let* $\mu \geq 0$ *with corresponding path* $p = SP(s,t,c,r,\mu)$, *and let* $z^* = \arg\min\{U(z) - \mu z\}$. *Then,* (p, z^*) *is a feasible solution w.r.t. constraint (5) if* $U'(r(p)) \leq \mu$.

Proof. Since $U(\cdot)$ is convex, $U(z) - \mu z$ is also convex and thus has a global optimum for z^* such that $U'(z^*) - \mu = 0$. The solution pair (p, z^*) is feasible if $r(p) \leq z^*$. If $U'(r(p)) < \mu = U'(z^*)$, then $r(p) < z^*$, since $U'(\cdot)$ is non-decreasing. If $U'(r(p)) = \mu = U'(z^*)$, then $r(p)$ minimizes $U(z) - \mu z$, and hence $r(p) \leq z^*$. ∎

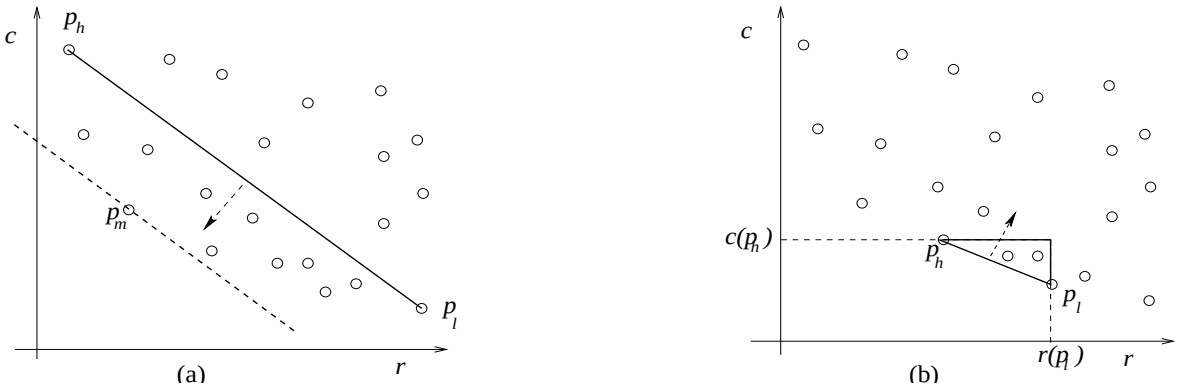

Figure 1: The algorithm. (a) First iteration. (b) Last iteration.

3.2 The Extended Hull Algorithm

We are now ready to give the algorithm for solving the relaxed INLP. We call a multiplier μ *feasible* if μ leads to a feasible solution pair (p, z); otherwise, we call μ *infeasible*.

During the process we maintain two multipliers μ_l and μ_h as well as the corresponding paths p_l and p_h. Multiplier μ_l is an infeasible one, while μ_h is feasible. Initially, $\mu_l = 0$ and the corresponding path p_l is the minimum cost path, while $\mu_h = +\infty$ and the corresponding path p_h is the minimum resource path. We adopt the geometric interpretation introduced in [9] and represent each path as a point in the r-c-plane.

In each iteration of the algorithm we find a path p_m corresponding to the multiplier $\mu_m = \frac{c(p_h)-c(p_l)}{r(p_l)-r(p_h)}$. This is equivalent to moving the line connecting p_h and p_l with slope $-\mu_m$ along its normal direction until we hit an extreme point in that direction (see Fig. 1(a)), i.e., a point belonging to the *lower convex hull* (that consists of those p for which there exists a μ such that $\forall q,\ c(p) + \mu r(p) \leq c(q) + \mu r(q)$). Multiplier μ_m leads to a solution (p_m, z_m). We check whether (p_m, z_m) is feasible using Lemma 3. This is equivalent to check whether $U'(r(p_m)) - \mu_m$ is positive, negative, or zero. In the latter case, we have found the optimal solution. If it is positive (infeasible solution), then by Lemma 2 we have to move to the "left" by setting p_l equal to p_m and μ_l to μ_m; otherwise, we have to move to the "right" by setting p_h equal to p_m and μ_h to μ_m.

We iterate this procedure as long as $\mu_l < \mu_m < \mu_h$, i.e., as long as we are able to find a new path below the line connecting the paths p_l and p_h. When such a path cannot be found, the algorithm stops (its stopping condition is met). As we shall see next, in this case either one of p_l and p_h is optimal, or there is a duality gap. We call the above the *extended hull algorithm*.

We are now ready to discuss the correctness of the algorithm. The following lemma implies that the paths p_l and p_h maintained during each iteration of the algorithm give us upper bounds on the resource and the cost of the optimal path.

Lemma 4 *Let $\mu \geq 0$ be a fixed multiplier and let $p = SP(s, t, c, r, \mu)$ be its corresponding path. Then, the following hold: (i) If $U'(r(p)) > \mu$, then $r(p)$ is an upper bound on the resource of the optimal path. (ii) If $U'(r(p)) < \mu$, then $c(p)$ is an upper bound on the cost of the optimal path. (iii) If $U'(r(p)) = \mu$, then p is optimal.*

Proof. (i) It suffices to prove that $\forall q \in P$ with $r(q) > r(p)$, it holds that $c(q) + U(r(q)) > c(p) + U(r(p))$. Since $U'(r(p)) > \mu$ and $U(\cdot)$ is convex, it holds that $U'(x) \geq U'(r(p)) > \mu$, $\forall x \geq r(p)$. Therefore, the function $U(x) - \mu x$ is increasing for $x \geq r(p)$, and thus we have that $\forall q \in P$ with $r(q) > r(p)$, $U(r(q)) - \mu r(q) > U(r(p)) - \mu r(p)$. By the definition of p we also have that $\forall q \in P$,

$$c(q) + \mu r(q) \geq c(p) + \mu r(p). \tag{9}$$

By adding the two inequalities we get $U(r(q)) + c(q) > U(r(p)) + c(p)$.

(ii) It suffices to prove that $\forall q \in P$ with $c(q) > c(p)$, it holds that $c(q)+U(r(q)) > c(p)+U(r(p))$. If $r(q) \geq r(p)$ the result is obvious. Therefore we only consider paths with $r(q) < r(p)$. Arguing as above we have that $U'(x) \leq U'(r(p)) < \mu$, $\forall x \leq r(p)$. Hence, the function $U(x) - \mu x$ is decreasing for $x \leq r(p)$, and thus we have that $\forall q \in P$ with $r(q) < r(p)$, $U(r(q)) - \mu r(q) > U(r(p)) - \mu r(p)$. Adding this last inequality to (9) we get $U(r(q)) + c(q) > U(r(p)) + c(p)$.

(iii) It suffices to prove that $\forall q \in P$: $c(p) + U(r(p)) \leq c(q) + U(r(q))$. Since function $U(x) - \mu x$ is convex and since $U'(r(p)) = \mu$, then $U(x) - \mu x$ is minimized for $x = r(p)$ and therefore $U(r(p)) - \mu r(p) \leq U(x) - \mu x$, $\forall x \in I\!R_0^+$. Adding this inequality for $x = r(q)$ to (9) completes the proof. ∎

It is not hard to see that if we have found a path which is shortest for both multipliers μ_l and μ_h, then it is shortest for any $\mu \in [\mu_l, \mu_h]$.

Lemma 5 *Given two fixed multipliers $0 < \mu_l < \mu_h$, such that for some path $p \in P$ and $\forall q \in P$ it holds that $c(p) + \mu_l r(p) \leq c(q) + \mu_l r(q)$ and $c(p) + \mu_h r(p) \leq c(q) + \mu_h r(q)$, then, $\forall \mu \in [\mu_l, \mu_h]$ it also holds that $c(p) + \mu r(p) \leq c(q) + \mu r(q)$, $\forall q \in P$.*

Proof. Obviously $\forall \mu \in [\mu_l, \mu_h]$, $\exists \theta \in [0, 1]$ (i.e., $\theta = \frac{\mu_h - \mu}{\mu_h - \mu_l}$) such that $\mu = \theta\mu_l + (1 - \theta)\mu_h$. Therefore, $\forall \mu \in [\mu_l, \mu_h]$ and $\forall q \in P$ $c(p) + \mu r(p) = (\theta + 1 - \theta)c(p) + (\theta\mu_l + (1 - \theta)\mu_h)r(p) = \theta(c(p) + \mu_l r(p)) + (1-\theta)(c(p) + \mu_h r(p)) \leq \theta(c(q) + \mu_l r(q)) + (1-\theta)(c(q) + \mu_h r(q)) = c(q) + \mu r(q)$. ∎

The next lemma (see Fig. 1(b)) completes the discussion on the correctness of the algorithm.

Lemma 6 *Let $0 < \mu_l < \mu_h$ be the multipliers at the end of the algorithm, $p_l = SP(s, t, c, r, \mu_l)$ and $p_h = SP(s, t, c, r, \mu_h)$ be their corresponding paths, and let $\mu_m = \frac{c(p_h) - c(p_l)}{r(p_l) - r(p_h)}$. (i) If $\mu_l \leq U'(r(p_l)) \leq \mu_m$, then p_l is optimal. (ii) If $\mu_m \leq U'(r(p_h)) \leq \mu_h$, then p_h is optimal. (iii) If $U'(r(p_h)) < \mu_m < U'(r(p_l))$, then there may be a duality gap. In this case, the optimum lies in the triangle defined by the lines $r = r(p_l)$, $c = c(p_h)$, and the line connecting paths p_l and p_h.*

Proof. Recall from the proof of Lemma 2 that $\mu_l \leq \mu_m = \frac{c(p_h) - c(p_l)}{r(p_l) - r(p_h)} \leq \mu_h$.
(i) If $\mu_l \leq U'(r(p_l)) \leq \mu_m$, from Lemma 5 we have that $\forall q \in P$: $c(p_l) + U'(r(p_l))r(p_l) \leq c(q) + U'(r(p_l))r(q)$. Thus, Lemma 4(iii) implies that p_l is optimal.
(ii) If $\mu_m \leq U'(r(p_h)) \leq \mu_h$, we have that $\forall q \in P$: $c(p_h) + U'(r(p_h))r(p_h) \leq c(q) + U'(r(p_h))r(q)$. Thus, Lemma 4(iii) implies that p_h is optimal.
(iii) Follows directly from Lemma 4 and the stopping condition of the algorithm. ∎

In the case where there is a duality gap, we can compute a lower bound to the actual optimum of the original INLP. We know that the optimum lies in the triangle defined by lines $r = r(p_l)$, $c = c(p_h)$, and the line connecting paths p_l and p_h. Let the latter line be $c + \mu r = W$, where $W = c(p_l) + \mu r(p_l) = c(p_h) + \mu r(p_h)$. In the best case the optimum path $(\overline{c}, \overline{r})$ would lie on the line connecting p_l and p_h, therefore it must be $\overline{c} + \mu\overline{r} = W$. A lower bound to the actual optimum is thus $\overline{c} + U(\overline{r})$, where $(\overline{c}, \overline{r})$ is the solution of the system of equations $\overline{c} + \mu\overline{r} = W$ and $U'(\overline{r}) = \mu$. The solution of the above system is $(\overline{c}, \overline{r}) = (W - \mu[U']^{-1}(\mu), [U']^{-1}(\mu))$, where $[U']^{-1}(\cdot)$ denotes the inverse function of $U'(\cdot)$ and provided that it exists. Therefore the lower bound is $\overline{c} + U(\overline{r}) = W - \mu[U']^{-1}(\mu) + U([U']^{-1}(\mu))$.

We now turn to the running time of the algorithm. The algorithm's progress can be measured by how much the area of the triangular unexplored region (defined by points p_l, p_h, and the intersection of the lines with slopes $-\mu_l$ and $-\mu_h$ passing through p_l and p_h, respectively) reduces at each iteration. We can prove (in a way similar to that in [9]) that the area reduces by at least 3/4.

Lemma 7 *The area of the unexplored region after an iteration of the extended hull algorithm is at most 1/4 of the area of the unexplored region after its immediately previous iteration.*

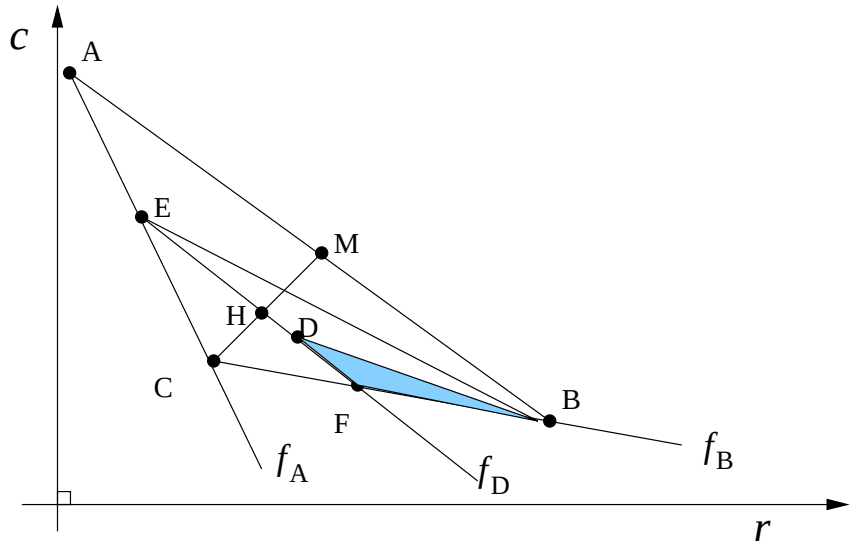

Figure 2: The change of the unexplored area after one iteration.

Proof. Let $\mathcal{E}_i$ and $\mathcal{E}_{i+1}$ denote the unexplored region after the i-th and the $i+1$-th iteration of the algorithm, respectively. Let $0 < \mu_l < \mu_h$ be the multipliers after the i-th iteration of the algorithm and let $B \equiv p_l$, and $A \equiv p_h$ be the points of the convex hull that correspond to them (see Fig 2). Let now f_A be the line with slope $-\mu_h$ that passes through A and f_B be the line of slope $-\mu_l$ that passes through B, and let C be the intersection point of f_A and f_B. During the iteration $i+1$, the algorithm examines the multiplier $\mu_m = \frac{c(p_h)-c(p_l)}{r(p_l)-r(p_h)}$, and let $D \equiv p_m$ be the corresponding point of the convex hull. Now, let f_D be the line with slope $-\mu_m$ that is parallel to the line segment AB and passes through D intersecting f_A and f_B in points E and F, respectively. Consider now the perpendicular from C to AB that intersects f_D in point H and AB in M. In the following, with $\mathcal{E}_X$ we denote the area of a region X.

The unexplored region after the i-th iteration is $\mathcal{E}_i = \mathcal{E}_{ABC}$. Assume w.l.o.g. that during the $i+1$-th iteration we move "to the right" (update a feasible solution). Then, the unexplored region after that iteration is the triangle DFB. Clearly, $\mathcal{E}_{i+1} = \mathcal{E}_{DFB} \leq \mathcal{E}_{EFB}$. Now, Thales' Theorem implies that $\frac{CF}{CB} = \frac{CE}{CA} = \frac{EF}{AB} = \frac{CH}{CM} = \delta$ and $\frac{HM}{CM} = 1-\delta$, where $\delta \in [0,1]$. Hence,

$$\begin{aligned}
\mathcal{E}_{EFB} &= \mathcal{E}_{AEFB} - \mathcal{E}_{AEB} \\
&= \frac{1}{2}(HM)((EF)+(AB)) - \frac{1}{2}(HM)(AB) \\
&= \frac{1}{2}(HM)(\delta+1)(AB) - \frac{1}{2}(HM)(AB) \\
&= \frac{1}{2}\delta(HM)(AB) \\
&= \delta(1-\delta)\frac{1}{2}(CM)(AB) \\
&= \delta(1-\delta)\mathcal{E}_{ABC}.
\end{aligned}$$

Therefore, $\frac{\mathcal{E}_{i+1}}{\mathcal{E}_i} \leq \frac{\mathcal{E}_{EFB}}{\mathcal{E}_{ABC}} = \delta(1-\delta)$. The quantity $\delta(1-\delta)$ is maximized for $\delta = \frac{1}{2}$. Consequently, $\frac{\mathcal{E}_{i+1}}{\mathcal{E}_i} \leq \frac{1}{4}$. ∎

For integral costs and resources in $[0, C]$ and $[0, R]$, respectively, the area of the initial triangle is at most $\frac{1}{2}(nR)(nC)$, the area of the minimum triangle is $\frac{1}{2}$, and by Lemma 7 the unexplored region is at least divided by four at each iteration. Thus, the algorithm ends after $O(\log(nRC))$

iterations. In each iteration a call to Dijkstra's algorithm is made. Hence, we have established the following.

Theorem 1 *The extended hull algorithm solves the Lagrangian relaxation of NASP in* $O(\log(nRC) \cdot (m + n \log n))$ *time.*

4 The Exact Algorithm

In this section, we describe an exact algorithm for solving NASP. The algorithm follows the general outline of the algorithms in [2, 6, 9] for solving RCSP and consists of three phases. In the first phase, we solve the relaxed problem using the extended hull algorithm (cf. Section 3). This provides us with a lower and an upper bound for the objective value as well as with an upper bound for the cost of the optimal path and an upper bound for the resource of the optimal path. In the second phase, we use the above cost and resource upper bounds to prune the graph by deleting nodes and edges that cannot lie on the optimal path. In the third phase, we close the gap between lower and upper bound and find the actual optimum by applying a path enumerating process in the pruned graph.

4.1 Improving the Efficiency of the Extended Hull Algorithm

To achieve better running times for phase 1, we develop two new heuristics that can be plugged in the extended hull algorithm. The first heuristic intends to reduce the number of iterations, accomplished by identifying cases where we can increase the algorithm's progress by selecting a multiplier other than that suggested by the hull approach. We refer to it as the *gradient* heuristic. The second heuristic aims at reducing the problem size by performing node reductions on the fly, using information obtained from previous iterations. We refer to it as the *on-line node reductions* heuristic.

Gradient Heuristic. Assume that there exists a multiplier μ' such that $\mu_l < \mu' < \mu_h$, and for which we can tell a priori whether the corresponding path is feasible or not (i.e., without computing it). We can then distinguish two cases in which we can examine μ' instead of μ_m, and achieve a better progress: (a) If μ' leads to an infeasible solution and $\mu_l < \mu_m < \mu'$, then Lemma 1 implies that μ_m also leads to an infeasible solution. Therefore, since both μ_m and μ' lead to an infeasible path, and μ' is closer to μ_h than μ_m, we can use μ' instead of μ_m in the update step and thus make a greater progress. (b) Similarly, if μ' leads to a feasible solution and $\mu' < \mu_m < \mu_h$, then μ_m also leads to a feasible solution. Again, we can use μ' instead of μ_m in the update step, since it is closer to μ_l. The following lemma suggests that when we are given a feasible (resp. infeasible) multiplier μ, we can always find an infeasible (resp. feasible) multiplier μ'.

Lemma 8 *Let* $\mu \geq 0$ *be a fixed multiplier,* $p = SP(s, t, c, r, \mu)$, $\mu' = U'(r(p))$, *and* $q = SP(s, t, c, r, \mu')$. *(i) If* $\mu' \leq \mu$, *then* $U'(r(q)) \geq \mu'$. *(ii) If* $\mu' \geq \mu$, *then* $U'(r(q)) \leq \mu'$.

Proof. (i) Since $U'(r(p)) \leq \mu$, we get from Lemma 1 that $r(q) \geq r(p)$. Since $U(\cdot)$ is convex, $U'(\cdot)$ is non-decreasing and thus $U'(r(q)) \geq U'(r(p))$. (ii) Follows similarly to (i). ∎

Lemma 8 along with the above discussion leads to the following heuristic that can be plugged in the extended hull algorithm. Let $0 \leq \mu_l \leq \mu_h$ be the multipliers at some iteration of the algorithm, and let p_l, p_h be the corresponding paths. If $U'(r(p_h)) \geq \frac{c(p_h)-c(p_l)}{r(p_l)-r(p_h)} \geq \mu_l$, then set $\mu_m = U'(r(p_h))$. If $U'(r(p_l)) \leq \frac{c(p_h)-c(p_l)}{r(p_l)-r(p_h)} \leq \mu_h$, then set $\mu_m = U'(r(p_l))$. Otherwise, set $\mu_m = \frac{c(p_h)-c(p_l)}{r(p_l)-r(p_h)}$.

On-line Node Reductions Heuristic. Further progress to the extended hull algorithm can be made by temporarily eliminating at each iteration those nodes that cannot lie on any path obtained by any subsequent iteration of the algorithm. These temporary eliminations of nodes are done in an on-line fashion and apply only to phase 1. The following lemma helps us to identify such nodes.

Lemma 9 *Let $p_l = SP(s,t,c,r,\mu_l)$ and $p_h = SP(s,t,c,r,\mu_h)$ be the paths corresponding to multipliers μ_l and μ_h, where $0 \leq \mu_l \leq \mu_h$. Let also $\mu_m = \frac{c(p_h)-c(p_l)}{r(p_l)-r(p_h)}$, and $B = c(p_l) + \mu_m r(p_l) = c(p_h) + \mu_m r(p_h)$. Then, $\forall \overline{\mu} \in [\mu_l, \mu_h]$ with corresponding path $\overline{p} = SP(s,t,c,r,\overline{\mu})$: (i) $c(\overline{p}) + \mu_m r(\overline{p}) \leq B$. (ii) $c(\overline{p}) + \mu_l r(\overline{p}) \leq c(p_h) + \mu_l r(p_h)$. (iii) $c(\overline{p}) + \mu_h r(\overline{p}) \leq c(p_l) + \mu_h r(p_l)$.*

Proof. By the definition of $\overline{p}$, we have $\forall q \in P$ that

$$c(\overline{p}) + \overline{\mu} r(\overline{p}) \leq c(q) + \overline{\mu} r(q) \tag{10}$$

(i) From inequality (10), it follows that $\forall q \in P$: $c(\overline{p}) + \overline{\mu} r(\overline{p}) + (\mu_m - \overline{\mu}) r(\overline{p}) \leq c(q) + \overline{\mu} r(q) + (\mu_m - \overline{\mu}) r(\overline{p})$ and therefore

$$c(\overline{p}) + \mu_m r(\overline{p}) \leq c(q) + \overline{\mu} r(q) + (\mu_m - \overline{\mu}) r(\overline{p}). \tag{11}$$

Since $\mu_l \leq \overline{\mu} \leq \mu_h$, we have that $r(p_l) \geq r(\overline{p}) \geq r(p_h)$. We consider two cases: (a) If $\mu_m \leq \overline{\mu}$, then by the fact that $r(\overline{p}) \leq r(p_l)$ inequality (11) becomes $c(\overline{p}) + \mu_m r(\overline{p}) \leq c(q) + \overline{\mu} r(q) + (\mu_m - \overline{\mu}) r(p_l) \leq c(p_l) + \mu_m r(p_l)$. (b) If $\mu_m > \overline{\mu}$, then by the fact that $r(\overline{p}) \geq r(p_h)$ inequality (11) becomes $c(\overline{p}) + \mu_m r(\overline{p}) \leq c(q) + \overline{\mu} r(q) + (\mu_m - \overline{\mu}) r(p_h) \leq c(p_h) + \mu_m r(p_h)$. This completes the proof of (i).
(ii) From inequality (10) we get that $\forall q \in P$: $c(\overline{p}) + \overline{\mu} r(\overline{p}) + (\mu_l - \overline{\mu}) r(\overline{p}) \leq c(q) + \overline{\mu} r(q) + (\mu_l - \overline{\mu}) r(\overline{p})$. Since $\mu_l \leq \overline{\mu}$ and $r(p_h) \leq r(\overline{p})$, we have that $c(\overline{p}) + \mu_l r(\overline{p}) \leq c(p_h) + \overline{\mu} r(p_h) + (\mu_l - \overline{\mu}) r(p_h) \leq c(p_h) + \mu_l r(p_h)$.
(iii) From inequality (10) we get that $\forall q \in P$: $c(\overline{p}) + \overline{\mu} r(\overline{p}) + (\mu_h - \overline{\mu}) r(\overline{p}) \leq c(q) + \overline{\mu} r(q) + (\mu_h - \overline{\mu}) r(\overline{p})$. Since $\mu_h \geq \overline{\mu}$ and $r(p_l) \geq r(\overline{p})$ we have that $c(\overline{p}) + \mu_h r(\overline{p}) \leq c(p_l) + \overline{\mu} r(p_l) + (\mu_h - \overline{\mu}) r(p_l) \leq c(p_l) + \mu_h r(p_l)$. ∎

Consider now an iteration of the algorithm with multipliers μ_l and μ_h and corresponding paths p_l and p_h. The multiplier tested during this and during any subsequent iteration is in the range $[\mu_l, \mu_h]$ and results in a path for which the statement of Lemma 9 applies. Let $d_\mu(x,y)$ denote a lower bound on the cost of a shortest x-y path w.r.t. the edge weights $w(\cdot) = c(\cdot) + \mu r(\cdot)$. Then, we can safely exclude from the graph any node u for which any of the following holds:

$$d_{\mu_m}(s,u) + d_{\mu_m}(u,t) > c(p_h) + \mu_m r(p_h) \tag{12}$$

$$d_{\mu_l}(s,u) + d_{\mu_l}(u,t) > c(p_h) + \mu_l r(p_h) \tag{13}$$

$$d_{\mu_h}(s,u) + d_{\mu_h}(u,t) > c(p_l) + \mu_h r(p_l) \tag{14}$$

To obtain the above lower bound distances, we need the following lemma.

Lemma 10 *Let $d_{\overline{\mu_l}}(u,v)$ and $d_{\overline{\mu_h}}(u,v)$ be lower bounds of the cost of a shortest u-v path w.r.t. edge weights $c(\cdot) + \overline{\mu_l} r(\cdot)$ and $c(\cdot) + \overline{\mu_h} r(\cdot)$, respectively, where $\overline{\mu_l} < \overline{\mu_h}$. Let $\overline{\mu} \in [\overline{\mu_l}, \overline{\mu_h}]$. Then, $d_{\overline{\mu}}(u,v) = \theta d_{\overline{\mu_l}}(u,v) + (1-\theta) d_{\overline{\mu_h}}(u,v)$ is also a lower bound of the cost of a shortest u-v path w.r.t. edge weights $c(\cdot) + \overline{\mu} r(\cdot)$, where $\theta = \frac{\overline{\mu_h} - \overline{\mu}}{\overline{\mu_h} - \overline{\mu_l}}$.*

Proof. By definition of $d_{\overline{\mu_l}}(u,v)$ and $d_{\overline{\mu_h}}(u,v)$, we have for any u-v path q that $d_{\overline{\mu_l}}(u,v) \leq c(q) + \overline{\mu_l} r(q)$ and $d_{\overline{\mu_h}}(u,v) \leq c(q) + \overline{\mu_h} r(q)$. Hence, for any such q, $d_{\overline{\mu}}(u,v) = \theta d_{\overline{\mu_l}}(u,v) + (1-\theta) d_{\overline{\mu_h}}(u,v) \leq \theta(c(q) + \overline{\mu_l} r(q)) + (1-\theta)(c(q) + \overline{\mu_h} r(q)) = c(q) + (\theta \overline{\mu_l} + (1-\theta)\overline{\mu_h}) r(q) = c(q) + \overline{\mu} r(q)$. ∎

To apply the on-line node reductions heuristic, we maintain a set R that consists of the nodes reduced until that iteration. Additionally, we maintain the lower bound distances $d_{\mu_l^s}(s,u)$, $d_{\mu_h^s}(s,u)$,

$d_{\mu_l^t}(u,t)$, $d_{\mu_h^t}(u,t)$, $\forall u \in V - R$, where $\mu_l^s \leq \mu_l$, $\mu_l^t \leq \mu_l$, $\mu_h^s \geq \mu_h$, $\mu_h^t \geq \mu_h$. These distances have been computed in previous shortest path computations (initially, they are set to zero). We call u *reduced* either if $u \in R$, or if any of the inequalities (12), (13), or (14) is satisfied.

In each iteration we test a multiplier μ_m by performing a forward (resp. backward)[2] run of Dikjstra's algorithm, using $w(\cdot) = c(\cdot) + \mu_m r(\cdot)$ as edge weights. For each node u extracted from the priority queue (used in Dijkstra's algorithm), we check whether u is reduced or not. If u is reduced but not in R, then we insert it to R. To check the inequalities, we need the lower bound distances in the LHS of (12), (13), and (14). To find these distances, we apply Lemma 10 using the known distances $d_{\mu_l^s}(s,u)$, $d_{\mu_h^s}(s,u)$, $d_{\mu_l^t}(u,t)$, and $d_{\mu_h^t}(u,t)$. If u is reduced, then we do not need to examine its outgoing (resp. incoming) edges. We stop Dijkstra's algorithm when the target (resp. source) node is settled. At this point, by setting the distances of the non-reduced unsettled nodes to the distance of the target (resp. source), we have new lower bound distance information, which can be used in subsequent iterations by updating the current information appropriately.

4.2 Graph Pruning

Recall that at the end of the extended hull algorithm, p_l gives an upper bound r_{max} on the resource and p_h gives an upper bound c_{max} on the cost of the optimal path. Using these bounds, we can eliminate those nodes and edges of the graph that lead to paths which lie outside the final triangle left by phase 1 (see Fig. 1(b)) and which we know that they are not optimal. Let c_{uv} (resp. r_{uv}) denote the cost (resp. resource) of the shortest u-v path w.r.t. cost (resp. resource). If for some node u we have that $c_{su} + c_{ut} > c_{max}$, or $r_{su} + r_{ut} > r_{max}$, then u can not lead to an optimal path. Similarly, if for some edge (u,v) we have that $c_{su} + c(u,v) + c_{vt} > c_{max}$, or $r_{su} + r(u,v) + r_{vt} > r_{max}$, then (u,v) can not lead to an optimal path. Thus, we can eliminate all such nodes and edges resulting in a reduced graph, and thus improving the running time of our algorithm. Finally, we apply the above idea recursively to the reduced graph and achieve further pruning.

4.3 Closing the Gap

To close the gap, we follow a label setting method that is essentially a modification of the one proposed in [9] for the RCSP problem. In the additive shortest path problem, Dijkstra's algorithm keeps a label for each node u representing the tentative distance of an s-u path. In our case each path p has a cost and a resource, and thus its corresponding label is a tuple (r_p, c_p). Consequently, with each node u a set of labels corresponding to s-u paths have to be stored, since now there is no total order among them. However, we can exploit the fact that some paths may *dominate* other paths, where a path p with label (r_p, c_p) dominates path q with label (r_q, c_q) if $c_p \leq c_q$ and $r_p \leq r_q$. Hence, at each node u we maintain a set $L(u)$ of undominated s-u paths, that is, every path in $L(u)$ does not dominate any other in the set. Recall the definitions of c_{uv}, r_{uv}, c_{max}, and r_{max} from Section 4.2. We call an s-u path p of cost c_p and resource r_p *non-promising*, if any of the following holds: (i) $c_p + c_{ut} > c_{max}$; (ii) $r_p + r_{ut} > r_{max}$; (iii) if there exists a Pareto-optimal u-t path q of cost c_q and resource r_q for which $c_p + c_q > c_{max}$ and $r_p + r_q > r_{max}$. Clearly, a non-promising path cannot be extended to an optimal one and hence it can be discarded. We would like to mention that

[2]Intuitively, we want the values of μ_l^s, μ_l^t and of μ_h^s, μ_h^t to be as close as possible to μ_l and μ_h, respectively, in order to get the best lower bound w.r.t. some $\mu \in [\mu_l, \mu_h]$. Hence, we can determine whether to do a forward or a backward run by checking which pair of multipliers is farther away from its target than the other. In particular, if $\mu_l^s < \mu_l^t$ and $\mu_h^s > \mu_h^t$, then perform a forward run; if $\mu_l^s > \mu_l^t$ and $\mu_h^s < \mu_h^t$, then perform a backward run; otherwise, choose arbitrarily. Now, if in addition we use the gradient heuristic, then we know a priori which one of μ_l and μ_h will be updated and thus we can further speed up the above tests.

in [9] only (i) and (ii) are checked. Our experiments revealed, however, that gap closing involving (iii) can be considerably faster (cf. Section 6).

Intuitively, the gap closing algorithm works as follows. It sweeps the line connecting p_l and p_h (see Fig. 1(b)) along its normal direction to hit all points in the triangle up to (and including) its right upper corner. Every time the line hits a point, the algorithm computes the value of the non-linear objective function, and at the end reports the point that gave the minimum value. As in Dijkstra's algorithm we use a priority queue Q to select promising paths that can be extended. The elements of Q are tuples of the form $(w_p, (r_p, c_p))$, where $w_p = c_p + \mu_m r_p$ is the combined cost of p and μ_m is the final multiplier of the extended hull algorithm. The elements of Q are ordered according to w_p. The use of Q has two advantages: (i) all paths extracted are Pareto-optimal; (ii) paths are examined in the order they are hit by sweeping the line connecting p_l and p_h along its normal direction. Initially, Q contains $(0, (0, 0))$ representing the trivial path from s to itself. While Q is not empty, we extract the minimum element corresponding to some s-u path p. If $u \equiv t$, we check whether p gives any improvement w.r.t. the s-t paths found so far; if $u \not\equiv t$, we try to extend it by an edge by considering all outgoing edges (u, v). For each such edge an s-v path is formed. If that path is a promising one, we check whether it is dominated by any path in $L(v)$. If indeed this is the case, then we discard it; otherwise, we insert it in $L(v)$ and in Q, and delete from both $L(v)$ and Q any labels (in $L(v)$) dominated by this one.

5 Generalization

The generalization of the objective function (4) involves two non-decreasing and convex functions $U_c : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ and $U_r : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ translating a path's cost (travel distance) and resource (travel time), respectively, to the common utility cost (money). We can also incorporate restrictions that may exist on the total cost or the resource of a path, thus adding constraints of the form $\sum_{p \in P} c_p x_p \leq B_c$ and $\sum_{p \in P} r_p x_p \leq B_r$, for some constants $B_c, B_r \geq 0$. Consequently, the resulting INLP is as follows

$$
\begin{aligned}
\min \quad & U_c \left(\sum_{p \in P} c_p x_p \right) + U_r \left(\sum_{p \in P} r_p x_p \right) \\
s.t. \quad & \sum_{p \in P} c_p x_p \leq B_c && (15) \\
& \sum_{p \in P} r_p x_p \leq B_r && (16) \\
& \sum_{p \in P} x_p = 1 \\
& x_p \in \{0, 1\}, \quad \forall p \in P
\end{aligned}
$$

and constitutes a generalization to both NASP and RCSP. We can show that the idea with the non-negative composite edge weights $w(\cdot) = c(\cdot) + \mu r(\cdot)$ (based on a Lagrangian multiplier μ) provides us with a polynomial algorithm for solving the relaxed version of the above INLP. To see that this is indeed possible, we provide generalized versions of the key Lemmata 4 and 6, which actually allow us to apply the extended hull algorithm and get the desired solution.

Lemma 11 *Let $\mu \geq 0$ be a fixed multiplier and let $p = SP(s, t, c, r, \mu)$ be its corresponding path. Then, the following hold: (i) If $U_r'(r(p)) > \mu U_c'(c(p))$, then $r(p)$ is an upper bound on the resource of the optimal path. (ii) If $U_r'(r(p)) < \mu U_c'(c(p))$, then $c(p)$ is an upper bound on the cost of the optimal path. (iii) If $U_r'(r(p)) = \mu U_c'(c(p))$, then p is optimal.*

Proof. Let $w(p) = c(p) + \mu r(p)$ be the composite cost of p. By definition of p, we have $\forall q \in P$

$$w(p) = c(p) + \mu r(p) \leq c(q) + \mu r(q). \tag{17}$$

Consider now the convex function $F(x) = U_r(x) + U_c(w(p) - \mu x)$, whose derivative is $F'(x) = U_r'(x) - \mu U_c'(w(p) - \mu x)$ and it is non-decreasing since $F(\cdot)$ is convex.

(i) It suffices to prove that $\forall q \in P$ with $r(q) > r(p)$, it holds that

$$U_c(c(q)) + U_r(r(q)) > U_c(c(p)) + U_r(r(p)).$$

If $U_r'(r(p)) > \mu U_c'(c(p))$, then $\forall x \geq r(p)$ we have that $F'(x) = U_r'(x) - \mu U_c'(w(p) - \mu x) \geq U_r'(r(p)) - \mu U_c'(w(p) - \mu r(p)) > 0$ and thus $F(x)$ is increasing for $x \geq r(p)$. Therefore, by (17) and $\forall q \in P$ with $r(q) > r(p)$, we get that

$$\begin{aligned} U_r(r(p)) + U_c(c(p)) &= U_r(r(p)) + U_c(w(p) - \mu r(p)) \\ &< U_r(r(q)) + U_c(w(p) - \mu r(q)) \\ &< U_r(r(q)) + U_c(c(q)) \end{aligned}$$

(ii) It suffices to prove that $\forall q \in P$ with $c(q) > c(p)$, it holds that $U_c(c(q)) + U_r(r(q)) > U_c(c(p)) + U_r(r(p))$. If $r(q) \geq r(p)$ the result is obvious. Therefore we only consider paths with $r(q) < r(p)$. Arguing as above, we see that if $U_r'(r(p)) < \mu U_c'(c(p))$, then $\forall x < r(p)$ we have that $F'(x) = U_r'(x) - \mu U_c'(w(p) - \mu x) \leq U_r'(r(p)) - \mu U_c'(w(p) - \mu r(p)) < 0$ and thus $F(x)$ is decreasing for $x \leq r(p)$. Consequently, $\forall q \in P$ with $r(q) < r(p)$ we get by (17) that $U_r(r(p)) + U_c(c(p)) = U_r(r(p)) + U_c(w(p) - \mu r(p)) < U_r(r(q)) + U_c(w(p) - \mu r(q)) < U_r(r(q)) + U_c(c(q))$.

(iii) It suffices to prove that $\forall q \in P$, it holds that

$$U_c(c(p)) + U_r(r(p)) \leq U_c(c(q)) + U_r(r(q)).$$

If $U_r'(r(p)) = \mu U_c'(c(p))$, then we have that $F'(r(p)) = U_r'(r(p)) - \mu U_c'(w(p) - \mu r(p)) = 0$. Since $F(x)$ is convex, it has a global minimum for $x = r(p)$ and thus $F(x) \geq F(r(p)), \forall x$. Therefore, $\forall q \in P$ we get

$$\begin{aligned} U_r(r(p)) + U_c(c(p)) &= U_r(r(p)) + U_c(w(p) - \mu r(p)) \\ &\leq U_r(r(q)) + U_c(w(p) - \mu r(q)) \\ &\leq U_r(r(q)) + U_c(c(q)) \end{aligned}$$

■

Lemma 12 *Let $0 < \mu_l < \mu_h$ be the multipliers at the end of the algorithm, $p_l = SP(s, t, c, r, \mu_l)$ and $p_h = SP(s, t, c, r, \mu_h)$ be their corresponding paths, and let $\mu_m = \frac{c(p_h) - c(p_l)}{r(p_l) - r(p_h)}$. (i) If $\mu_l \leq \frac{U_r'(r(p_l))}{U_c'(c(p_l))} \leq \mu_m$ and p_l is feasible w.r.t. constraints* (15) & (16)*, then it is also optimal. (ii) If $\mu_m \leq \frac{U_r'(r(p_h))}{U_c'(c(p_h))} \leq \mu_h$ and p_h is feasible w.r.t. constraints* (15) & (16)*, then it is also optimal. (iii) If $\frac{U_r'(r(p_h))}{U_c'(c(p_h))} < \mu_m < \frac{U_r'(r(p_l))}{U_c'(c(p_l))}$, then there may be a duality gap. In this case, the optimum lies in the triangle defined by the lines $r = r(p_l)$, $c = c(p_h)$, and the line connecting paths p_l and p_h.*

Proof. Recall from the proof of Lemma 2 that $\mu_l \leq \mu_m = \frac{c(p_h) - c(p_l)}{r(p_l) - r(p_h)} \leq \mu_h$.

(i) If $\mu_l \leq \frac{U_r'(r(p_l))}{U_c'(c(p_l))} \leq \mu_m$, from Lemma 5 we have that $\forall q \in P$: $c(p_l) + \frac{U_r'(r(p_l))}{U_c'(c(p_l))} r(p_l) \leq c(q) + \frac{U_r'(r(p_l))}{U_c'(c(p_l))} r(q)$. Thus, since p_l is feasible, Lemma 11(iii) implies that it is also optimal.

(ii) If $\mu_m \leq \frac{U_r'(r(p_h))}{U_c'(c(p_h))} \leq \mu_h$, from Lemma 5 we have that $\forall q \in P$: $c(p_h) + \frac{U_r'(r(p_h))}{U_c'(c(p_h))} r(p_h) \leq c(q) + \frac{U_r'(r(p_h))}{U_c'(c(p_h))} r(q)$. Thus, since p_h is feasible, Lemma 11(iii) implies that it is also optimal.
(iii) Follows directly from Lemma 11 and the stopping condition of the algorithm. ∎

We are now ready to describe the modified extended hull algorithm for solving the relaxed problem of the generalized version. We say that a multiplier μ leads to a *cost-feasible* path $p = SP(s,t,c,r,\mu)$ if the condition $(c(p) \leq B_c$ **and** $(r(p) > B_r$ **or** $U_r'(r(p)) \geq \mu U_c'(c(p))))$ is satisfied. Similarly, we say that a multiplier μ leads to a *resource-feasible* path $p = SP(s,t,c,r,\mu)$ if the condition $(r(p) \leq B_r$ **and** $(c(p) > B_c$ **or** $U_r'(r(p)) \leq \mu U_c'(c(p))))$ is satisfied. During the process we maintain two multipliers μ_l and μ_h as well as their corresponding paths p_l and p_h, such that μ_l leads to a cost-feasible path, while μ_h leads to a resource-feasible path. Initially, $\mu_l = 0$ and the corresponding path p_l is the minimum cost path; if that path violates the cost bound B_c, then the problem is clearly infeasible and the algorithm stops. Similarly, $\mu_h = +\infty$ and the corresponding path p_h is the minimum resource path; if that path violates the resource bound B_r, then the problem is infeasible and the algorithm stops. In each iteration of the algorithm we find a path p_m corresponding to the multiplier $\mu_m = \frac{c(p_h)-c(p_l)}{r(p_l)-r(p_h)}$. If p_m is cost-feasible, then we set p_l to p_m and μ_l to μ_m. If p_m is resource-feasible, then we set p_h to p_m and μ_h to μ_m. If p_m is neither cost-feasible nor resource-feasible, then it must violate both cost and resource bounds B_c and B_r and the algorithm stops, since in that case the problem is infeasible. We iterate this procedure as long as $\mu_l < \mu_m < \mu_h$. i.e., as long as we are able to find a new path below the line connecting the paths p_l and p_h. When no new path can be found, the algorithm stops. By Lemma 12, either p_l or p_h is optimal, or a duality gap exists. We can also obtain a lower bound, in a way similar to that in Section 3.2.

The running time of the modified extended hull algorithm is the same with that of Theorem 1. The exact algorithm described in Section 4 can be applied in this case, too. All heuristics but the gradient remain unchanged. The gradient heuristic has to be modified according to the following lemma which is a generalization of Lemma 8.

Lemma 13 *Let $\mu \geq 0$ be a fixed multiplier with corresponding path $p = SP(s,t,c,r,\mu)$, and let $\mu' = \frac{U_r'(r(p))}{U_c'(c(p))}$ with corresponding path $q = SP(s,t,c,r,\mu')$. Then: (i) If $\mu' \leq \mu$, then $U_r'(r(q)) \geq \mu' U_c'(c(q))$. (ii) If $\mu' \geq \mu$, then $U_r'(r(q)) \leq \mu' U_c'(c(q))$.*

Proof. (i) Since $\mu' \leq \mu$, we get from Lemma 1 that $r(q) \geq r(p)$ and $c(q) \leq c(p)$. Since $U_r(\cdot)$ and $U_c(\cdot)$ are convex, $U_r'(\cdot)$ and $U_c'(\cdot)$ are non-decreasing and thus $U_r'(r(q)) \geq U_r'(r(p))$ and $U_c'(c(p)) \geq U_c'(c(q))$. Multiplying these two inequalities we get $U_r'(r(q))U_c'(c(p)) \geq U_r'(r(p))U_c'(c(q))$ and therefore $U_r'(r(q)) \geq \frac{U_r'(r(p))}{U_c'(c(p))} U_c'(c(q)) = \mu' U_c'(c(q))$.
(ii) Since $\mu' \geq \mu$, we get from Lemma 1 that $r(q) \leq r(p)$ and $c(q) \geq c(p)$, and consequently $U_r'(r(q)) \leq U_r'(r(p))$ and $U_c'(c(p)) \leq U_c'(c(q))$ (by the non-decreasingness of the first derivatives). Multiplying these two inequalities we get $U_r'(r(q))U_c'(c(p)) \leq U_r'(r(p))U_c'(c(q))$ and therefore $U_r'(r(q)) \leq \frac{U_r'(r(p))}{U_c'(c(p))} U_c'(c(q)) = \mu' U_c'(c(q))$. ∎

6 Experiments

We conducted a series of experiments based on several implementations[3] of the extended hull algorithm and of the exact algorithm with the heuristics described in Section 4. In particular, for the solution of the Lagrangian relaxation of NASP (phase 1 of the exact algorithm), we have implemented the extended hull algorithm (EHA) of Section 3.2 (using Dijkstra's algorithm for shortest

[3]Implementations were in `C++` using LEDA. Experiments were carried out in a SunFire 280R with an UltraSPARC-III processor at 750 MHz and 512 MB of memory.

path computations and aborting when the target node is found), the variant of the hull approach described in [10] using the bidirectional Dijkstra's algorithm (MW), and EHA using different heuristics: EHA[B] (with the bidirectional Dijkstra's algorithm), EHA[G] (with the gradient heuristic), EHA[NR] (with the on-line node reductions heuristic), EHA[G+NR] (gradient and on-line node reductions heuristics), and EHA[G+B] (gradient heuristic with bidirectional Dijkstra).

With this bulk of implementations, we performed experiments on synthetic and real world data.

Synthetic Data. We consider random grid graphs and digital elevation models (DEM) – following the experimental set-up in [9]. For grid graphs, we used integral edge costs and resources chosen randomly and independently from $[100, 200]$. A DEM is a grid graph where each node has an associated height value. We used integral values as heights chosen randomly and independently from $[100, 200]$. The same applies for the edge resources. The cost of an edge was the absolute difference between the heights of its endpoints. In both types of input, we used the objective function (4) which is the most commonly used in transportation applications [4, 5, 12] with $U(r)$ quadratic and appropriately scaled; that is, the objective function was $\frac{c}{\delta_c(s,t)} + (\frac{r}{\delta_r(s,t)})^2$, where by $\delta_\lambda(s,t)$ we denote the weight of the shortest s-t path with respect to the weight function $\lambda(\cdot)$.

The experimental results on grid graphs and DEMs are reported in Tables 1 and 2. We report on time performances (secs) and on the number of shortest path computations (#SP). The reported values are averages over several query (s,t) pairs that involve either border nodes of the grid or random pairs. In the former case, we always include the top-left and the bottom-right corner to the queries.

$N \times N$ Grid – 100 Border (s,t) pairs										
N	50		100		200		400		600	
Implem.	#SP	Time	#SP	Time	#SP	Time	#SP	Time	#SP	Time
MW	9.35	0.125	10.9	0.650	12.3	4.862	14.8	33.540	15.5	91.640
EHA	6.74	0.095	7.61	0.490	8.31	3.359	9.68	22.128	9.97	61.627
EHA[B]	6.74	0.088	7.61	0.444	8.31	3.211	9.68	21.648	9.97	57.902
EHA[G]	4.35	0.061	4.14	0.260	4.38	1.721	4.73	10.609	4.62	27.965
EHA[NR]	6.74	0.063	7.61	0.281	8.31	1.683	9.68	9.6354	9.97	25.354
EHA[G+NR]	4.35	0.053	4.14	0.234	4.38	1.451	4.73	8.3676	4.62	21.879
EHA[G+B]	4.35	0.054	4.14	0.233	4.38	1.650	4.73	10.437	4.62	26.290

$N \times N$ DEM – 100 Border (s,t) pairs										
N	50		100		200		400		600	
Implem.	#SP	Time	#SP	Time	#SP	Time	#SP	Time	#SP	Time
MW	9.55	0.127	10.8	0.651	12.2	4.896	14.3	31.50	15.3	89.3267
EHA	7.24	0.105	8.04	0.530	8.69	3.561	9.88	22.27	10.5	64.2516
EHA[B]	7.24	0.093	8.04	0.474	8.69	3.407	9.88	21.27	10.5	61.0365
EHA[G]	4.26	0.061	4.25	0.275	4.10	1.635	4.39	9.669	4.65	27.7620
EHA[NR]	7.24	0.075	8.05	0.331	8.69	1.957	9.88	10.64	10.5	28.5771
EHA[G+NR]	4.26	0.054	4.25	0.246	4.10	1.466	4.39	8.070	4.65	21.7356
EHA[G+B]	4.26	0.053	4.25	0.242	4.10	1.566	4.39	9.270	4.65	26.4830

Table 1: Nodes s and t are chosen randomly between nodes at the left and the right border of the grid, resp.

For the solution of the relaxed problem (phase 1 of the exact algorithm), we observe that MW, despite the fact that it is implemented using bidirectional Dijkstra (which is usually faster than pure Dijkstra), has an inferior performance w.r.t. both EHA and EHA[B]. This is due to the fact that it performs much more shortest path computations, since MW does not use derivatives. Regarding the heuristics, we observe that both gradient and on-line node reductions are rather powerful heuristics. In all cases, the gradient heuristic reduces the number of shortest path computations from 20% to

$N \times N$ Grid – 500 Random (s,t) pairs										
N	50		100		200		400		600	
Implem.	#SP	Time	#SP	Time	#SP	Time	#SP	Time	#SP	Time
MW	7.13	0.053	8.90	0.306	10.8	2.311	12.6	15.28	13.3	40.18
EHA	5.53	0.049	6.50	0.259	7.58	1.861	8.53	11.62	8.90	31.12
EHA[B]	5.53	0.040	6.50	0.217	7.58	1.609	8.53	10.04	8.90	26.01
EHA[G]	4.32	0.035	4.30	0.155	4.27	0.957	4.36	5.611	4.45	14.72
EHA[NR]	5.53	0.032	6.50	0.145	7.58	0.874	8.53	4.769	8.90	11.98
EHA[G+NR]	4.32	0.029	4.30	0.129	4.27	0.787	4.36	4.335	4.45	10.98
EHA[G+B]	4.32	0.028	4.30	0.127	4.27	0.815	4.36	4.831	4.45	12.19

$N \times N$ DEM – 500 Random (s,t) pairs										
N	50		100		200		400		600	
Implem.	#SP	Time	#SP	Time	#SP	Time	#SP	Time	#SP	Time
MW	7.51	0.058	9.16	0.328	10.7	2.23	12.4	14.29	13.4	41.57
EHA	5.96	0.055	6.88	0.289	7.82	1.89	8.75	11.50	9.35	34.33
EHA[B]	5.96	0.044	6.88	0.238	7.82	1.60	8.75	9.790	9.35	28.26
EHA[G]	4.50	0.038	4.31	0.163	4.25	0.93	4.27	5.179	4.34	14.92
EHA[NR]	5.96	0.037	6.88	0.167	7.82	0.93	8.75	4.932	9.35	13.49
EHA[G+NR]	4.50	0.031	4.31	0.138	4.25	0.78	4.27	4.134	4.34	11.30
EHA[G+B]	4.50	0.029	4.31	0.131	4.25	0.77	4.27	4.376	4.34	12.17

Table 2: Nodes s and t were chosen randomly among all nodes.

60%, and the improvement is more evident as the graph size gets larger. The on-line node reductions heuristic achieves in almost all cases a better running time than the gradient heuristic. The most interesting characteristic of this heuristic is that its performance increases with the number of shortest path computations. This is due to the successive reduction of the problem size. Note that this heuristic can be applied to other related problems (e.g., RCSP), where we expect to improve their running times, too. Finally, the combination of both heuristics gives further improvements and makes EHA[G+NR] along with EHA[G+B] to be the fastest implementations of the exact algorithm.

We close by commenting on pruning and gap closing (phases 2 and 3 of the exact algorithm). Our experiments revealed that these two phases were performed only for a very small number of instances, actually less than 2%; see Table 3. This implies that for the 98% of the instances tested, the reported times in Tables 1 and 2 essentially provide the total running time of the exact algorithm. The specific running times of pruning and gap closing, for those instances that these phases had to be carried out, are given in Table 3. We made two implementations of gap closing: the first is the one described in [9] (gc[MZ]), while the second is our method described in Section 4.3 (gc[new]) that uses Pareto path information[4]. The Pareto paths were obtained by a backward Dijkstra's run using the composite weights $w(\cdot) = c(\cdot) + \mu_m r(\cdot)$, where μ_m is the final value of the multiplier of phase 1. For small graph instances, the two implementations are close to each other. As the graph gets larger, however, gc[new] performs better, since much more paths are discarded and thus less delete-min operations are executed.

Real Road Networks. We also conducted experiments on real road networks from USA and specifically on two high-detail state road networks of Florida and Alabama, and on the NHPN (National Highway Planning Network) covering the entire continental US. The data sets of the state networks consist of four levels of road networks (interstate highways, principal arterial roads, major arterial roads, and rural minor arterial roads). The Florida high-detail road network consists of 50109 nodes and 133134 edges, the Alabama high-detail road network consists of 66082 nodes

[4]We do not report on the k-shortest path approach [6, 7], since gc[MZ] was faster as it is also confirmed in [9].

	$N \times N$ Grid – 10000 Border (s,t) pairs				$N \times N$ DEM – 10000 Border (s,t) pairs			
N	100		200		100		200	
#GCP	101		182		78		187	
Phase	Time	#del_min	Time	#del_min	Time	#del_min	Time	#del_min
pruning	0.267	–	2.209	–	0.342	–	2.062	–
gc[MZ]	0.359	14260.9	32.420	744056	0.416	16574.8	11.877	359077
gc[new]	0.276	9459.82	17.440	456400	0.401	13279.2	7.057	217919

	$N \times N$ Grid – 10000 Random (s,t) pairs				$N \times N$ DEM – 10000 Random (s,t) pairs			
N	100		200		100		200	
#GCP	171		171		139		168	
Phase	Time	#del_min	Time	#del_min	Time	#del_min	Time	#del_min
pruning	0.158	–	0.823	–	0.191	–	1.281	–
gc[MZ]	0.302	10499.7	3.507	98876.5	0.170	4622.12	2.123	69093.1
gc[new]	0.285	8739.01	2.522	64586.8	0.182	3578.34	1.837	48338.7

Table 3: By #GPC we denote the number of instances ((s,t) pairs) that needed pruning and gap closing.

and 185986 edges, and the US NHPN consists of 75417 nodes and 205998 edges. These particular data sets were also used in [13].

Road Networks – 1000 Random (s,t) pairs						
Input Graph	U.S. NHPN		Florida		Alabama	
Implem.	#SP	Time	#SP	Time	#SP	Time
MW	10.58	3.43	10.19	2.05	9.88	2.45
EHA	7.34	2.44	7.10	1.33	6.88	2.14
EHA[B]	7.34	2.32	7.10	1.45	6.88	1.70
EHA[G]	4.01	1.23	4.01	0.69	3.98	1.15
EHA[NR]	7.34	1.11	7.10	0.63	6.88	0.97
EHA[G+NR]	4.01	1.04	4.01	0.60	3.98	0.95
EHA[G+B]	4.01	1.14	4.01	0.73	3.98	0.89

Road Networks – 10000 Random (s,t) pairs						
Graph	U.S. NHPN		Florida		Alabama	
#GCP	110		51		74	
Phase	Time	#del_min	Time	#del_min	Time	#del_min
pruning	0.92	–	0.43	–	0.74	–
gc[MZ]	5.64	201156	1.46	37284	1.34	32940
gc[new]	4.68	144848	0.94	30889	1.30	29620

Table 4: Nodes s and t are chosen randomly among all nodes.

In all cases the cost and the resource of an edge were taken equal to the actual distance of its endpoints (provided with the data) multiplied by a uniform random variable in $[1,2]$ (to differentiate the cost from the resource value), and subsequently scaled and rounded appropriately so that the resulting numbers are integers in $[0,10000]$. The non-linear objective function considered in all cases was the same as that for grid graphs and DEMs.

The experimental results (see Table 4) are similar to those obtained using grid graphs and DEMs. Once again MW has an inferior performance w.r.t. the other implementations. Regarding the heuristics, we observe the same behavior as in the other experiments, and the fastest implementations are again EHA[G+B] and EHA[G+NR]. We also observe similar behavior between the two implementations for the gap-closing phase, where gc[new] performs slightly better.

References

[1] R. Ahuja, T. Magnanti, and J. Orlin, *Network Flows*, Prentice-Hall, 1993.

[2] J. Beasley and N. Christofides, "An Algorithm for the Resource Constrained Shortest Path Problem", *Networks* 19(1989), pp. 379-394.

[3] E.W. Cheney, *Introduction to Approximation Theory*, McGraw-Hill, New York, 1966.

[4] S. Gabriel and D. Bernstein, "The Traffic Equilibrium Problem with Nonadditive Path Costs", *Transportation Science* 31:4(1997), pp. 337-348.

[5] S. Gabriel and D. Bernstein, "Nonadditive Shortest Paths: Subproblems in Multi-Agent Competitive Network Models", *Computational & Mathematical Organization Theory* 6(2000), pp. 29-45.

[6] G. Handler and I. Zang, "A Dual Algorithm for the Constrained Shortest Path Problem", *Networks* 10(1980), pp. 293-310.

[7] M. Henig, "The Shortest Path Problem with Two Objective Functions", *European Journal of Operational Research* 25(1985), pp. 281-291.

[8] D. Hensen and T. Truong, "Valuation of Travel Times Savings", *Journal of Transport Economics and Policy* (1985), pp. 237-260.

[9] K. Mehlhorn and M. Ziegelmann, "Resource Constrained Shortest Paths", in *Algorithms* – ESA 2000, Lecture Notes in Computer Science 1879 (Springer-Verlag 2000), pp. 326-337.

[10] P. Mirchandani and M. Wiecek, "Routing with Nonlinear Multiattribute Cost Functions", *Applied Mathematics and Computation* 54(1993), pp. 215-239.

[11] C. Papadimitriou and M. Yannakakis, "On the Approximability of Trade-offs and Optimal Access of Web Sources", in *Proc. 41st Symp. on Foundations of Computer Science* – FOCS 2000, pp. 86-92.

[12] K. Scott and D. Bernstein, "Solving a Best Path Problem when the Value of Time Function is Nonlinear", preprint 980976 of the *Annual Meeting of the Transportation Research Board*, 1997.

[13] F.B. Zhan and C.E. Noon, "Shortest Path Algorithms: An Evaluation using Real Road Networks", *Transportation Science* 32:1(1998), pp. 65-73.