Non-Additive Shortest Paths*

George Tsaggouris[1,2] Christos Zaroliagis[1,2]

[1] Computer Technology Institute, P.O. Box 1122, 26110 Patras, Greece
[2] Department of Computer Engineering and Informatics,
University of Patras, 26500 Patras, Greece
{tsaggour,zaro}@ceid.upatras.gr

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 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 a fundamental problem in network optimization encompassing a vast number of applications. Given a digraph 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. 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

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

optimal path both w.r.t. travel cost and time. 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 (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.

In this paper, we concentrate on devising an exact algorithm for solving NASP as well as on computing a Lagrangian relaxation of 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)[3] 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

[3] RCSP asks for finding a minimum cost path whose resource does not exceed a specific bound. Costs and resources are *additive* along paths. RCSP is modelled as an ILP.

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. 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 a linear program (RCSP) to a method that solves the relaxation of a 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. 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.

Proofs and parts omitted due to space limitations can be found in [13].

2 Preliminaries and Problem Formulation

In the following, let $G = (V, E)$ be a digraph with n nodes and m edges, and let $c : E \to \mathbb{R}_0^+$ 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.

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

$$s.t. \sum_{p \in P} x_p = 1 \quad ; \quad x_p \in \{0, 1\}, \forall p \in P \tag{2}$$

Suppose now that, in addition to the cost function c, there is also a resource function $r : E \to \mathbb{R}_0^+$ associating each edge $e \in E$ with a non-negative resource

$r(e)$. 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 an non-decreasing and convex function $U : \mathbb{R}_0^+ \to \mathbb{R}_0^+$ with first derivative $U' : \mathbb{R}_0^+ \to \mathbb{R}_0^+$. 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:

$$\min \sum_{p\in P} c_p x_p + U(\sum_{p\in P} r_p x_p) \tag{3}$$

Substituting (1) with the new objective function (3) and keeping the same constraints (2), 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. 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, since it is the most commonly used in applications (see e.g., [4, 5, 12]), and it better introduces our approach.

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 $U(\cdot)$ is a non-decreasing function, we can relax this equality constraint to inequality, and get the following INLP:

$$\begin{aligned} \min \ & \sum_{p\in P} c_p x_p + U(z) \\ s.t. \ & \sum_{p\in P} r_p x_p \leq z \\ & \sum_{p\in P} x_p = 1 \quad ; \quad x_p \in \{0,1\}, \forall p \in P \end{aligned} \tag{4}$$

We can relax the above using Lagrangian relaxation: introduce a Lagrangian multiplier $\mu \geq 0$, multiply the two sides of (4) by μ, and turn constraint (4) in the objective function. The objective now becomes $L(\mu) = \min \sum_{p\in P} (c_p + \mu r_p) x_p + U(z) - \mu z$, while constraints (2) remain unchanged. $L(\mu)$ is a lower bound on the optimal value of the objective function (3) for any value of the Lagrangian multiplier μ [1, Ch. 16]. A best lower bound can be found by solving the problem $\max_{\mu\geq 0} L(\mu)$ (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

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

$$\begin{aligned} \text{where} \qquad L_p(\mu) &= \min \sum_{p \in P} (c_p + \mu r_p) x_p \\ & \qquad s.t. \sum_{p \in P} x_p = 1 \quad ; \quad x_p \in \{0,1\}, \forall p \in P \\ \text{and} \qquad L_z(\mu) &= \min \; U(z) - \mu z \end{aligned} \tag{5}$$

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. 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* $0 \le \mu_l \le \mu_h$. *Then, (i)* $r(p_l) \ge r(p_h)$*; (ii)* $c(p_l) \le c(p_h)$*; (iii)* $r(p_l) - z_l \ge r(p_h) - z_h$*; and (iv)* $c(p_l) + U(z_l) \le c(p_h) + U(z_h)$.

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 (4). 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 (4) is violated.

The next two lemmata allow the effective extension of the hull algorithm 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* $p_l = SP(s, t, c, r, \mu_l)$, $p_h = SP(s, t, c, r, \mu_h)$, $z_l = \arg\min\{U(z) - \mu_l z\}$, $z_h = \arg\min\{U(z) - \mu_h z\}$, $p_m = SP(s, t, c, r, \mu_m)$, $z_m = \arg\min\{U(z) - \mu_m z\}$, $0 \le \mu_l \le \mu_h$, *and* $\mu_m = \frac{c(p_h) - c(p_l)}{r(p_l) - r(p_h)}$. *Then, (i)* $r(p_l) - z_l \ge r(p_m) - z_m \ge r(p_h) - z_h$*, and (ii)* $c(p_l) + U(z_l) \le c(p_m) + U(z_m) \le c(p_h) + U(z_h)$.

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 (5) 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 (4) if $U'(r(p)) \leq \mu$.*

3.2 The Extended Hull Algorithm

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

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

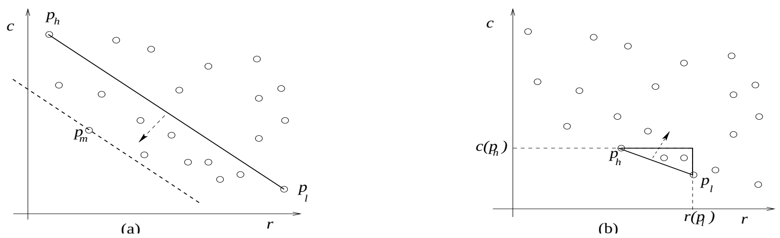

Fig. 1. The algorithm. (a) First iteration. (b) Last iteration.

Lemma 5. *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* p_l *and* p_h*.*

Proof. From Lemma 2, we have that $\mu_l \leq \mu_m = \frac{c(p_h)-c(p_l)}{r(p_l)-r(p_h)} \leq \mu_h$. We will also use the fact (cf. [13]) that if we have found a path which is shortest for both multipliers μ_l and μ_h, then it is shortest for any $\mu \in [\mu_l, \mu_h]$.
(i) If $\mu_l \leq U'(r(p_l)) \leq \mu_m$, from the above fact 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) Similar to (i). (iii) Follows directly from Lemma 4 and the stopping condition of the algorithm. □

In the case where there is a duality gap (see Fig. 1(b)), we can compute a lower bound to the actual optimum of the original INLP. Let the line connecting p_l and p_h 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 $(\overline{c}, \overline{r})$ lies on that line and 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 $(\overline{c}, \overline{r}) = (W - \mu[U']^{-1}(\mu), [U']^{-1}(\mu))$ is the solution of the system of equations $\overline{c} + \mu\overline{r} = W$ and $U'(\overline{r}) = \mu$ (here $[U']^{-1}(\cdot)$ denotes the inverse function of $U'(\cdot)$).

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]; cf. [13]) that the area reduces by at least $3/4$. 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 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* $O(\log(nRC)(m + n\log n))$ *time.*

4 The Exact Algorithm

The exact algorithm for solving NASP 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. This provides us with a lower and an upper bound for the objective value as well as with an upper bound on the cost and the resource of the optimal path. In the second phase, we use these cost and resource upper bounds to prune the graph by deleting nodes and edges that cannot lie on the optimal path (details for this phase can be found in [13]). 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'. Its proof follows by Lemma 1 and the convexity of $U(\cdot)$.

Lemma 6. *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 $U'(r(q)) \leq \mu'$.*

Lemma 6 along with the above discussion lead 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. 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:

$$d_{\mu_m}(s, u) + d_{\mu_m}(u, t) > c(p_h) + \mu_m r(p_h) \tag{6}$$

$$d_{\mu_l}(s, u) + d_{\mu_l}(u, t) > c(p_h) + \mu_l r(p_h) \tag{7}$$

$$d_{\mu_h}(s, u) + d_{\mu_h}(u, t) > c(p_l) + \mu_h r(p_l) \tag{8}$$

The above lower bound distances can be found using the following lemma.

Lemma 7. *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 $\theta = \frac{\overline{\mu_h} - \overline{\mu}}{\overline{\mu_h} - \overline{\mu_l}}$.*

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 (6), (7), or (8) is satisfied.

In each iteration we test a multiplier μ_m by performing a forward (resp. backward) run of Dikjstra's algorithm [13], 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 (6),(7), and (8), which can be found by applying Lemma 7 on 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 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. Let c_{uv} (resp. r_{uv}) denote the cost (resp. resource) of the shortest u-v path w.r.t. cost (resp. resource), and let c_{max} (resp. r_{max}) be the upper bound on the cost (resp. resource) of the optimal path returned by phase 1. 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 in [9] only (i) and (ii) are checked. Our experiments revealed, however, that gap closing involving (iii) can be considerably faster (cf. Section 6). The gap closing algorithm is implemented with the use of a priority queue, whose elements 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 phase 1. Paths are examined in the order they are hit by sweeping the line connecting p_l and p_h along its normal direction.

5 Generalization

The generalization of the objective function (3) 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

$$\begin{aligned} \min \;\; & U_c\left(\sum_{p \in P} c_p x_p\right) + U_r\left(\sum_{p \in P} r_p x_p\right) \\ s.t. \;\; & \sum_{p \in P} c_p x_p \leq B_c \quad ; \quad \sum_{p \in P} r_p x_p \leq B_r \\ & \sum_{p \in P} x_p = 1 \quad ; \quad x_p \in \{0,1\}, \;\; \forall p \in P \end{aligned} \tag{9}$$

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. We provide generalized versions of the key Lemmata 4 and 5 (see [13]), which actually allow us to apply the hull approach and get the desired solution. The generalized versions of these Lemmata result in certain modifications of the extended hull algorithm (see [13]), without sacrificing its worst-case complexity.

6 Experiments

We conducted a series of experiments based on several implementations[4] 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 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 random grid graphs and digital elevation models (DEM) – following the experimental set-up in [9] – as well as on real-world data based on US road networks. 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 (3) 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 are reported in Table 1 (similar results hold for DEMs; cf. [13]). 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 border nodes of the grid (similar results hold for randomly chosen pairs).

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 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 re-

[4] 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.

duction of the problem size. 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. Regarding 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 lower part of Table 1. This implies that for the 98% of the instances tested, the time bounds of Table 1 (upper part) essentially provide the total running time of the exact algorithm. 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 4.2 (gc[new]) that uses Pareto path information. The larger the graph, the better the performance of gc[new], since much more paths are discarded and thus less delete-min operations are executed.

$N \times N$ 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

	$N \times N$ Grid – 10000 Border (s,t) pairs			
N	100		200	
#GCP	101		182	
Phase	Time	#del_min	Time	#del_min
pruning	0.267	–	2.209	–
gc[MZ]	0.359	14260.9	32.420	744056
gc[new]	0.276	9459.8	17.440	456400

Table 1. #GCP denotes the number of (s,t) pairs that needed pruning and gap closing.

Regarding the experiments on real road networks from USA, we consider the high-detail state road networks of Florida and Alabama, and 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 and 185986 edges, and the US NHPN consists of 75417 nodes and 205998 edges. These particular data sets were also used in [14]. 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 2) are similar to those obtained using grid graphs and DEMs.

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 2. Nodes s and t are chosen randomly among all nodes.

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, NY, 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", *Comp. & Math. Organiz. Theory* 6 (2000).
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, LNCS 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 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 *Transport. Research Board*, 1997.
13. G. Tsaggouris and C. Zaroliagis, "Non-Additive Shortest Paths", Tech. Report TR-2004/03/01, Computer Technology Institute, Patras, March 2004.
14. F.B. Zhan and C.E. Noon, "Shortest Path Algorithms: An Evaluation using Real Road Networks", *Transportation Science* 32:1(1998), pp. 65-73.