Automatic Decrease of the Penalty Parameter in Exact Penalty Function Methods

Marcel Mongeau
Department of Mathematics and Statistics
University of Edinburgh, EH9 3JZ, U. K.

Annick Sartenaer
Department of Mathematics
Facultés Universitaires ND de la Paix, Namur, Belgium

Abstract

This paper presents an analysis of the involvement of the penalty parameter in exact penalty function methods that yields modifications to the standard outer loop which decreases the penalty parameter (typically dividing it by a constant). The procedure presented is based on the simple idea of making explicit the dependence of the penalty function upon the penalty parameter and is illustrated on a linear programming problem with the l_1 exact penalty function and an active-set approach. The procedure decreases the penalty parameter, when needed, to the *maximal* value allowing the inner minimization algorithm to leave the current iterate. It moreover avoids unnecessary calculations in the iteration following the step in which the penalty parameter is decreased. We report on preliminary computational results which show that this method can require fewer iterations than the standard way to update the penalty parameter. This approach permits a better understanding of the performance of exact penalty methods.

Key words: Exact penalty method, penalty parameter, linear programming, active-set approach.

Abbreviated title for use as running head: Automatic Decrease of the Penalty Parameter.

1 Introduction

Exact penalty functions are used to provide a transformation of a constrained optimization problem to an unconstrained optimization problem. Indeed, one approach to solving the constrained problem

$$\begin{array}{rrcll} \min\limits_{x \in I\!\!R^n} & f(x) & & & \\ \text{subject to} & f_i(x) & = & 0, & i \in E \\ & f_i(x) & \geq & 0, & i \in I, \end{array} \tag{1}$$

where the functions f and $\{f_i\}_{i \in E \cup I}$ are continuously differentiable and the index sets E and I are finite and disjoint, is to construct an *exact penalty function*. That is, a function f_γ minimized locally by a solution x^* to the above constrained problem, provided $\gamma > 0$ is suitably chosen. The most popular nondifferentiable exact penalty function is the *absolute value* or l_1 *exact penalty function*:

$$f_\gamma(x) \equiv \gamma f(x) + \sum_{i \in E} |f_i(x)| - \sum_{i \in I} \min[0, f_i(x)], \quad \gamma > 0. \tag{2}$$

Let us consider the case where the functions f and $\{f_i\}_{i \in E \cup I}$ are convex. Under mild conditions (see [10], page 143), there is a threshold value $\bar{\gamma} > 0$ such that for any $0 < \gamma < \bar{\gamma}$, an unconstrained local minimum of f_γ is a local minimum for (1). The reader is referred to [9] for the more general non-convex case.

Coleman and Conn gave further justification for the design of algorithms based on the optimization of an exact penalty function. In [5], the similarity shown between the optimality conditions for the exact penalty function and those for problem (1) reinforces the idea that the two problems are closely related.

We wish to minimize, as a subproblem, the nondifferentiable function f_γ for a given *penalty parameter* value $\gamma > 0$. Let us denote $f'(\cdot; d)$ the directional derivative of f in the direction $d \in I\!\!R^n$, defined as

$$f'(x; d) \equiv \lim_{t \to 0^+} \frac{f(x + td) - f(x)}{t}$$

when this limit exists. A point x^* is termed a *first-order minimizer* of the exact penalty function f_γ when, for all directions $d \in I\!\!R^n$, we have

$$f'_\gamma(x^*; d) \geq 0.$$

Note that this directional derivative exists by properties of (2).

Once we find out that one of the two following situations holds, then we decrease γ:

Situation 1: The current iterate is a local optimum of f_γ which is infeasible with respect to the original problem.

Situation 2: We have found an infeasible ray with respect to the original problem along which f_γ is decreasing.

We decrease γ in these cases because we know that for a positive γ value small enough, a first-order minimizer of f_γ is in general a Kuhn-Tucker point of the original constrained problem.

Advantages of the penalty approach are that it does not require a feasible point to begin and *best* solutions can be determined when no feasible point exists, best in the sense of: the l_1 norm of the infeasibilities is minimized. *Exact* penalty functions have moreover the feature of not introducing ill-conditioned Hessian matrices, since the penalty parameter does not need to be made arbitrarily small to achieve convergence to a solution of the original problem.

Typically, whenever it is necessary to decrease the penalty parameter γ in exact penalty function methods, rather arbitrary techniques are used. As pointed out by Gamble, Conn and Pulleyblank in [12], very little theory has been developed in this area. Since one can consider that γ determines the relative weight between the objective function f and the constraint violations, its value is rather significant.

In this paper we shall introduce modifications to the standard outer loop used in exact penalty function techniques that typically reduces γ by a constant factor, so that first, we shall avoid doing unnecessary calculations in an iteration following the step in which we decrease the penalty parameter γ, and secondly, when decreasing γ we shall attempt to do so "wisely". Indeed, even though, in theory, exact penalty methods allow to solve the problem in a single iteration for a penalty parameter below the threshold value, in practice, it can be more efficient to allow steps out of the feasible region, using a decreasing sequence of penalty parameters starting above the threshold value. Instead of decreasing γ by some arbitrary factor (possibly many times) until some change occurs, we shall rather, quite easily, determine the *maximal* value γ' such that a change takes place. We then decrease the penalty parameter to any value $\gamma < \gamma'$ to leave situation 1 or 2 described above.

The theoretical aspects of the modifications to the exact penalty method we present in this paper will be illustrated for the sake of simplicity on a linear programming problem with an algorithm using the l_1 exact penalty function and an active-set approach. Nevertheless, we may expect these ideas to be exploited analogously in the framework of more general nonlinear problems. This is due to the fact that the results presented are a straightforward consequence of the definition of an *exact* penalty function and follow from the simple idea of making explicit the dependence of f_γ upon the penalty parameter γ in the unconstrained subproblem: $\min_{x \in \mathbb{R}^n} f_\gamma(x)$.

We first describe, in section 2, a typical algorithm to minimize the l_1 exact penalty function for the linear programming problem. In section 3, we then illustrate our method to decrease the penalty parameter on this algorithm. We discuss, in section 4, the practical application of our procedure in the light of computational experimentation on a minimum-cost network flow problem. We finally conclude in section 5.

2 An l_1 Exact Penalty Function Algorithm for the Linear Programming Problem

Consider the l_1 exact penalty function:

$$f_\gamma(x) \equiv \gamma c^T x + \sum_{i \in E} |a_i^T x - b_i| - \sum_{i \in I} \min[0, a_i^T x - b_i], \tag{3}$$

corresponding to the problem:

$$\begin{array}{rrcll} \min\limits_{x \in I\!\!R^n} & c^T x & & & \\ \text{subject to} & a_i^T x & = & b_i, & i \in E \\ & a_i^T x & \geq & b_i, & i \in I, \end{array} \tag{4}$$

where c and a_i, $i \in E \cup I$, are vectors of $I\!\!R^n$ and $b_i \in I\!\!R$, $i \in E \cup I$. The index sets E and I are finite and disjoint.

Let $\mathcal{A}(y)$, or simply $\mathcal{A}$ when it is clear from the context which is the current point, be the index set corresponding to the constraints which are *active* at y, i.e.

$$\mathcal{A}(y) \equiv \{i \in E \cup I : a_i^T x = b_i\}.$$

Consider the following function, which is differentiable in a neighbourhood of $y \in I\!\!R^n$:

$$\varphi_{\gamma,y}(x) \equiv \gamma c^T x + \sum_{i \in E \setminus \mathcal{A}(y)} |a_i^T x - b_i| - \sum_{i \in I \setminus \mathcal{A}(y)} \min[0, a_i^T x - b_i].$$

Clearly,

$$f_\gamma(x) = \varphi_{\gamma,y}(x) + \sum_{i \in E \cap \mathcal{A}(y)} |a_i^T x - b_i| - \sum_{i \in I \cap \mathcal{A}(y)} \min[0, a_i^T x - b_i],$$

and $\nabla \varphi_{\gamma,y}(y)$ is what Bartels, Conn and Sinclair [3] called the *restricted gradient* of f_γ at y. The restricted gradient, denoted $\nabla \varphi_\gamma(y)$ for simplicity, is the gradient of the restriction of f_γ to the space $\mathcal{N}(A^T(y))$, where $\mathcal{N}(A^T(y))$ denotes the null space of the matrix $A^T(y)$ having as its rows the vectors $\{a_i\}_{i \in \mathcal{A}(y)}$. If the columns of a matrix Z form a basis of $\mathcal{N}(A^T(y))$, then $Z^T \nabla \varphi_\gamma(y)$ is the *reduced restricted gradient* of f_γ at y.

It can be shown (see for instance [5]) that assuming $\{a_i\}_{i \in \mathcal{A}(x^*)}$ are linearly independent, the following are necessary and sufficient conditions for x^* to be a minimizer of f_γ: there exist scalars $\{u_\gamma^i\}_{i \in \mathcal{A}(x^*)}$ such that

Condition 1:

$$\nabla \varphi_\gamma(x^*) = \sum_{i \in \mathcal{A}(x^*)} u_\gamma^i a_i, \quad \text{or equivalently} \quad Z^T \nabla \varphi_\gamma(x^*) = 0;$$

Condition 2:

$$\text{(i)} \qquad -1 \le u_\gamma^i \le 1, \;\; \text{for all } \; i \in E \cap \mathcal{A}(x^*)$$

and

$$\text{(ii)} \qquad 0 \le u_\gamma^i \le 1, \;\; \text{for all } \; i \in I \cap \mathcal{A}(x^*).$$

Let us interpret these conditions. We say that a direction $d^i \in I\!\!R^n$ is a direction *dropping activity* i alone if $a_j^T d^i = 0$ for all $j \in \mathcal{A}(x^*) \setminus \{i\}$. In the case where condition 1 is satisfied, one can verify that if $d^{i^+} \in I\!\!R^n$ is a direction dropping activity i such that $a_i^T d^{i^+} > 0$, then

$$(u_\gamma^i + 1)|a_i^T d^{i^+}| \;\text{ and }\; u_\gamma^i |a_i^T d^{i^+}|$$

are the directional derivatives of f_γ at x^* in the direction d^{i^+}, for $i \in E$ and $i \in I$ respectively. In the same way, if $d^{i^-} \in I\!\!R^n$ is a direction dropping activity i such that $a_i^T d^{i^-} < 0$, then

$$(-u_\gamma^i + 1)|a_i^T d^{i^-}|$$

is the directional derivative in the direction d^{i^-} for $i \in E \cup I$. Thus, the above conditions 1 and 2 are saying that x^* is a minimizer of f_γ if and only if the reduced restricted gradient is null and f_γ is non-decreasing along each of the $2|\mathcal{A}(x^*)|$ $(\le 2n)$ possible single-dropping directions—there is one such direction corresponding to each of the *multiplier rules* of condition 2.

We now present an iterative algorithm for solving problem (4) via an exact penalty function method based on the above optimality conditions. At iteration k, we thus want to solve the unconstrained subproblem $\min_x f_{\gamma^k}(x)$, where γ^k is the current penalty parameter value. Let x^k be the current iterate. Define P to be the orthogonal projector onto the space $\mathcal{N}(A^T(x^k))$ and P_{-j} to be the orthogonal projector onto the space orthogonal to the space spanned by $\{a_i\}_{i \in \mathcal{A}(x^k) \setminus \{j\}}$, where $j \in \mathcal{A}(x^k)$. These projectors are computed with a suitable factorization of the matrix of activities, $A(x^k)$, updated from one iteration to another.

The algorithm can be summarized as follows: if the reduced restricted gradient of f_{γ^k} at x^k, $Z^T \nabla \varphi_{\gamma^k}(x^k)$, is non-null, then

$$d^k \equiv -P(\nabla \varphi_{\gamma^k}(x^k))$$

is a descent direction for f_{γ^k}. Otherwise, we compute the coefficients $\{u_{\gamma^k}^i\}_{i \in \mathcal{A}(x^k)}$ satisfying condition 1 above by solving a least-squares problem. Note that since the reduced restricted gradient is zero, the least-squares problem has an exact solution (zero residuals). If there exists $j \in \mathcal{A}(x^k)$ such that $u_{\gamma^k}^j < -1$, $j \in E$ or $u_{\gamma^k}^j < 0$, $j \in I$, then we can take $d^k \equiv P_{-j}(a_j)$ as a dropping descent direction. We can take $d^k \equiv -P_{-j}(a_j)$ if there exists $j \in \mathcal{A}(x^k)$ such that $u_{\gamma^k}^j > 1$, $j \in E \cup I$. When we find a descent direction d^k, we then determine a step size by solving the univariate piecewise-linear problem: $\min_{\alpha > 0} f_{\gamma^k}(x^k + \alpha d^k)$, moving from one breakpoint of f_{γ^k} to the next, in the direction d^k.

Unless the penalty function is unbounded, the value of f_{γ^k} will start increasing. We then stop and obtain a new iterate. When we reach an iterate x^k which is a minimizer of f_{γ^k} but which is also infeasible, we first perform an iteration of the algorithm with $\gamma = 0$ to determine whether x^k is a minimizer of f_γ for $\gamma = \gamma^k$ *and* for $\gamma = 0$. In such a case, we establish the infeasibility of the original problem without the need to decrease the penalty parameter several times to reach this conclusion.

In this context, the situations in the presence of which we decrease the penalty parameter are:

Situation 1': The current iterate is infeasible and it is a minimizer of f_{γ^k} but it is not a minimizer of f_γ for $\gamma = 0$.

Situation 2': We have found an infeasible ray with respect to the original problem along which f_γ is decreasing.

The flow chart of this algorithm, used as the inner algorithm with a standard way to decrease the penalty parameter, is displayed in figure 1.

Note finally that an easy generalization of a result in linear programming (lemma 5.1 in [2]) that was first proved in the case of linear networks (see [12]) can improve this algorithm. We however do not develop it here in order to simplify the presentation of our procedure to decrease the penalty parameter (a version of our procedure in which this improvement is integrated can be found in [16] and in [17]).

3 Decrease of the Penalty Parameter

We consider a new subproblem each time we are in one of situations 1' or 2', since we are required to decrease the value of the penalty parameter γ.

By definition of an *exact* penalty function, there exists a threshold value $\bar{\gamma} > 0$ such that Kuhn-Tucker points of the original constrained optimization problem (4) are first-order minimizers of the penalty function f_γ given by (3), *for all* $\gamma < \bar{\gamma}$ [11]. As we are considering the case where f_γ is convex, we can leave out the "first-order" terminology and talk simply about *minimizers* of f_γ. Thus, we can deduce the following necessary condition for the current iterate, x^k, to be a Kuhn-Tucker point of our original problem and the current penalty parameter value, γ^k, to be less than or equal to the threshold value $\bar{\gamma}$:

Condition 0: x^k is a minimizer of f_γ, for all $\gamma \leq \gamma^k$.

In other words, if a point x^* is a solution of our original problem, we know that the necessary and sufficient conditions 1 and 2 mentioned above for x^* to be a minimizer of f_γ must hold *for all* $\gamma < \bar{\gamma}$, for a certain $\bar{\gamma} > 0$. We shall elaborate an automatic way to decrease the penalty parameter from examining what can be done when condition 0 is not satisfied.

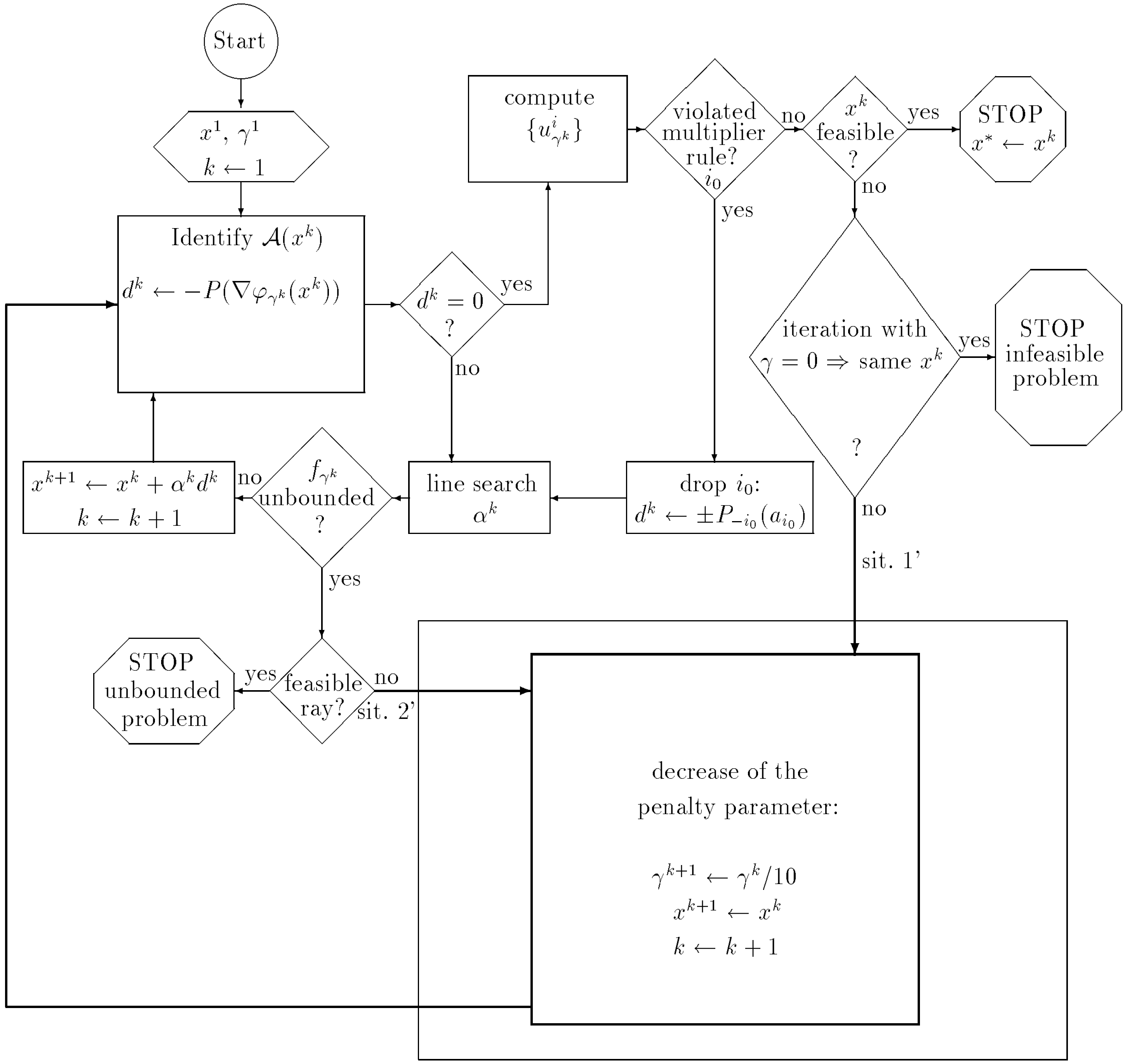

Figure 1: The inner loop algorithm with a standard way to decrease γ

Let us now look into the algorithm outlined in section 2 while emphasizing the dependence of f_γ on γ. Thus, we make explicit the contribution of the original objective function and that of the penalty terms of f_γ.

We know we can write the penalty function as

$$f_\gamma(x) = \gamma c^T x + \hat{f}(x),$$

where

$$\hat{f}(x) \equiv \sum_{i \in E} |a_i^T x - b_i| - \sum_{i \in I} \min[0, a_i^T x - b_i].$$

We decompose φ_{γ,x^k} similarly:

$$\varphi_{\gamma,x^k}(x) = \gamma c^T x + \hat{\varphi}_{x^k}(x), \tag{5}$$

where

$$\hat{\varphi}_{x^k}(x) \equiv \sum_{i \in E \setminus \mathcal{A}(x^k)} |a_i^T x - b_i| - \sum_{i \in I \setminus \mathcal{A}(x^k)} \min[0, a_i^T x - b_i].$$

We will denote $\hat{\varphi}_{x^k}(x)$ simply by $\hat{\varphi}(x)$ for notational ease, whenever no ambiguity is introduced. So the '^' parts correspond to the penalty part of the penalty function.

At each iteration, we modify the algorithm presented in section 2 in the following manner: instead of computing $P(\nabla\varphi_{\gamma^k}(x^k))$, we compute both $P(c)$ and $P(\nabla\hat{\varphi}(x^k))$. Note that when $P(\nabla\varphi_{\gamma^k}(x^k))$ is needed, it is readily computed using (5).

From condition 0, it is clear that if x^k were a Kuhn-Tucker point of the original constrained problem, then we must have

$$P(\nabla\varphi_\gamma(x^k)) = 0, \quad \text{for all } \gamma \leq \gamma^k$$

or simply $P(c) = 0$. This is a consequence of the fact that x^k feasible implies $P(\nabla\varphi_{\gamma^k}(x^k)) = \gamma^k P(c)$. Thus, if in the course of the algorithm we obtain an iterate x^k such that

Situation 0': $P(\nabla\varphi_{\gamma^k}(x^k)) = 0$ but $P(c)$, or equivalently $P(\nabla\hat{\varphi}(x^k))$, is non-null,

then we see immediately that x^k cannot be a Kuhn-Tucker point of the original constrained problem or γ^k is above the threshold value.

Situation 0' implies that if γ^k is decreased to *any* $\gamma' < \gamma^k$, x^k will no longer satisfy condition 1 for $f_{\gamma'}$, since

$$P(\nabla\varphi_{\gamma'}(x^k)) = \gamma' P(c) + P(\nabla\hat{\varphi}(x^k)) \neq 0. \tag{6}$$

Whence $-P(\nabla\varphi_{\gamma'}(x^k))$ is a straightforward descent direction. This descent direction is obtained at no extra cost using (6), since $P(c)$ and $P(\nabla\hat{\varphi}(x^k))$ were already computed. Situation 0' is thus the first situation where an automatic decrease of the penalty parameter can take place.

Let us henceforward assume that at every iterate x^k satisfying condition 1 the gradients of the activities are linearly independent. Another modification to the algorithm outlined in section 2 is that, instead of solving only one least-squares problem to obtain the coefficients $\{u^i_{\gamma^k}\}_{i\in\mathcal{A}(x^k)}$, we solve two such problems to compute $\{u^i\}_{i\in\mathcal{A}(x^k)}$ and $\{\hat{u}^i\}_{i\in\mathcal{A}(x^k)}$, the coefficients of $\{a_i\}_{i\in\mathcal{A}(x^k)}$ in the linear combinations of c and $\nabla\hat{\varphi}(x^k)$, respectively, in terms of the columns of $A(x^k)$. Clearly,

$$u^i_{\gamma^k} = \gamma^k u^i + \hat{u}^i,$$

for all $i \in \mathcal{A}(x^k)$ and the same matrix factorization is used in solving both least-squares problems.

By looking how condition 0 is violated in situation 1', the next situation in the presence of which we have to decrease the penalty parameter, we deduce how γ^k might then be decreased. Situation 1' occurs when the current iterate, x^k, is infeasible with respect to the original problem, is a minimizer of f_{γ^k} but is not a minimizer of f_γ with $\gamma = 0$. Assuming we are not in situation 0', i.e. $P(c) = P(\nabla\hat{\varphi}(x^k)) = 0$, it is condition 2 that must be violated for f_γ with $\gamma = 0$. That is to say, either there exists $i \in E \cap \mathcal{A}(x^k)$ such that $-1 \leq \hat{u}^i \leq 1$ is violated or there exists $i \in I \cap \mathcal{A}(x^k)$ such that $0 \leq \hat{u}^i \leq 1$ is violated. Let us suppose, for example, that the violated condition above corresponds to having an $i \in E \cap \mathcal{A}(x^k)$ such that $\hat{u}^i < -1$. Thus, decreasing the penalty parameter, we shall hit a value $\gamma' > 0$ such that x^k does not satisfy condition 2, for $f_{\gamma'}$, any more. Indeed, the following part of condition 2 part (i):

$$u^i_\gamma \equiv \gamma u^i + \hat{u}^i \geq -1,$$

is violated for all $\gamma < -(\hat{u}^i + 1)/u^i$, where $u^i > 0$ for all $i \in E \cap \mathcal{A}(x^k)$ such that $\hat{u}^i < -1$, since $u^i_{\gamma^k} \geq -1$ for all $i \in E \cap \mathcal{A}(x^k)$ as x^k is a minimizer of f_{γ^k}. Analogously for the other inequalities in condition 2, we have that x^k is not a minimizer of f_γ, for all $\gamma < \gamma'$, where

$$\gamma' \equiv \max\left(\max_{i\in I\cap\mathcal{A}(x^k):\hat{u}^i<0} -\frac{\hat{u}^i}{u^i}, \max_{i\in E\cap\mathcal{A}(x^k):\hat{u}^i<-1} -\frac{\hat{u}^i+1}{u^i}, \max_{i\in\mathcal{A}(x^k):\hat{u}^i>1} -\frac{\hat{u}^i-1}{u^i}\right). \qquad (7)$$

Let i_0 be the index $i \in \mathcal{A}(x^k)$ corresponding to the maximum in (7). Thus, we decrease the penalty parameter to some value $\gamma < \gamma'$, and we know without any further calculations that in the next iteration,

$$d^{k+1} \equiv P_{-i_0}(a_{i_0}) \text{ or } d^{k+1} \equiv -P_{-i_0}(a_{i_0})$$

will be a descent direction for f_γ, for any $\gamma < \gamma'$.

The last case in which we decrease the penalty parameter is situation 2': we obtain an infeasible ray with respect to the original problem along which f_{γ^k} is decreasing. Let d^k be the current descent direction along which we are doing the line search. The fact that we are in situation 2' means that $f'_{\gamma^k}(\cdot; d^k)$ is negative at each breakpoint, i.e.

$$f'_{\gamma^k}(x^k + \alpha_j d^k; d^k) < 0, \; 0 \leq j \leq m, \qquad (8)$$

where $\{\alpha_j\}_{0\le j\le m}$ are the step sizes corresponding to the breakpoints encountered along the search direction. If we decrease the penalty parameter, we shall hit a value γ' such that for some $1 \le j_0 \le m$, we have

$$f'_{\gamma'}(x^k + \alpha_{j_0} d^k; d^k) = 0.$$

Since

$$f'_{\gamma'}(x^k + \alpha_{j_0} d^k; d^k) = \gamma' c^T d^k + \hat{f}'(x^k + \alpha_{j_0} d^k; d^k), \tag{9}$$

the maximal such value of the penalty parameter is given by:

$$\gamma' \equiv \max_{0\le j\le m} \left\{ \frac{-\hat{f}'(x^k + \alpha_j d^k; d^k)}{c^T d^k} \right\}. \tag{10}$$

Note that this value for γ' is well-defined, non-negative and strictly smaller than γ^k, since at least for the last breakpoint, i.e. for $j = m$,

$$\hat{f}'(x^k + \alpha_j d^k; d^k) \ge 0 \ \text{ and } \ c^T d^k < 0,$$

as the penalty part $\hat{f}$ is bounded below and (8) holds for $j = m$.

The value of $\hat{f}'(\cdot; d^k)$ can be readily computed from the known entities $\{a_i\}_{i\in E\cup I}$ and d^k (and $\hat{u}^{i_0}$ when d^k is a direction dropping activity i_0).

Let j_0 be the smallest index for which the maximum is attained in (10). Since f_γ is convex, the maximum is necessarily attained for $j = m$, though it might be attained as well for some $j < m$. We then set $\gamma^{k+1} \equiv \gamma'$ given by (10). Note that this value is non-null if we assume the original problem to be feasible and bounded below. Indeed, for $\gamma' = 0$ in (10), we would have $\hat{f}'(x^k + \alpha_{j_0} d^k; d^k) = 0$. But one could then show that d^k is a feasible ray, from any feasible point, such that $c^T d^k < 0$. This contradicts the hypothesis that the original problem is bounded.

We then distinguish two possibilities:

a) If $j_0 \ne 0$, then we know by convexity, without any need to solve any new least-squares problems and to project again, that d^k is a descent direction for $f_{\gamma^{k+1}}$, as it was one for f_{γ^k}. We know moreover, without going through a new line search, that

$$x^{k+1} \equiv x^k + \alpha_{j_0} d^k$$

will be the next iterate.

b) Otherwise, it means that d^k is not a descent direction any more for $f_{\gamma^{k+1}}$. In the case where condition 1 was satisfied for x^k and f_{γ^k}, we can straight away verify whether condition 2 is satisfied for $x^{k+1}(= x^k)$ and $f_{\gamma^{k+1}}$, as $\{u^i\}_{i\in\mathcal{A}}$ and $\{\hat{u}^i\}_{i\in\mathcal{A}}$ are already known. We can thereby find out whether we can drop an activity to get a descent direction. If condition 1 was not satisfied for x^k and f_{γ^k}, then we must be in situation 0' and we thus decrease again the penalty parameter accordingly, as seen before.

Finally note that with the above modification, it is not necessary to actually perform the iteration with $\gamma = 0$, since with $\gamma = 0$ we have $u^i_\gamma = \hat{u}^i$, whose value is already known at each iteration.

To summarize, the penalty algorithm using the automatic decrease of the penalty parameter follows.

Penalty Algorithm

Step 0: [Initialization]
Choose any $x^1 \in I\!R^n$, $\epsilon > 0$ and $\gamma^1 > 0$. Set $k \leftarrow 1$.
Step 1: [Inner Loop]
Minimize f_{γ^k} (see sections 2 and 3).
Step 2: [Update of the Penalty Parameter]
If situation 0', then $\gamma^{k+1} = \gamma^k(1-\epsilon)$,
$x^{k+1} = x^k$, $d^k = -[\gamma^{k+1}P(c) + P(\nabla\hat{\varphi}(x^k))]$, $k \leftarrow k+1$ and go to step 1.
If situation 1', then $\gamma^{k+1} = \gamma'(1-\epsilon)$ given by (7),
$x^{k+1} = x^k$, $d^{k+1} = \pm P_{-i_0}(a_{i_0})$, $k \leftarrow k+1$ and go to step 1.
If situation 2', then $\gamma^{k+1} = \gamma'(1-\epsilon)$ given by (10).
· If $j_0 \neq 0$, then $x^{k+1} = x^k + \alpha_{j_0} d^k$, $k \leftarrow k+1$ and go to step 1.
· Otherwise, $x^{k+1} = x^k$, $k \leftarrow k+1$ and go to step 1.

Note that "go to step 1" stands for the specific branching, within the inner-loop unconstrained minimization of f_{γ^k}, which is displayed by the flow chart of figure 2.

Theorem 1 (Convergence) *Suppose that problem (4) is bounded below and feasible. Assume moreover that the vectors $\{a_i\}_{i\in\mathcal{A}(x^k)}$ are linearly independent at each iterate x^k encountered in the course of the algorithm.*

Then the penalty algorithm converges globally (i.e. from any starting point) in a finite number of iterations.

Proof: The exactness of the penalty function f_γ (see [10] theorem 40) together with the fact that $\gamma^{k+1} \leq \gamma^k(1-\epsilon)$, with $\epsilon > 0$, for every k, imply that we need to consider only a finite number of values of γ before reaching a penalty parameter value $\bar{\gamma} > 0$ such that an unconstrained minimizer of $f_{\bar{\gamma}}$ corresponds to a minimizer of the original constrained problem. The proof follows by the finiteness of the unconstrained minimization algorithm used in step 1 of the penalty algorithm. The algorithm described in section 2 is, for instance, a specialization of the piecewise-linear minimization algorithm described in [7]. The reader is referred to the finite-step convergence theorem therein. $\Box$

Figure 2 gives a flow chart of the algorithm of section 2 when implemented with our procedure to decrease the penalty parameter. Numbers in parentheses refer to the labels of the equations within the text. Note that *dead* $\leftarrow$ *true* means that condition 1 is satisfied at the current iterate.

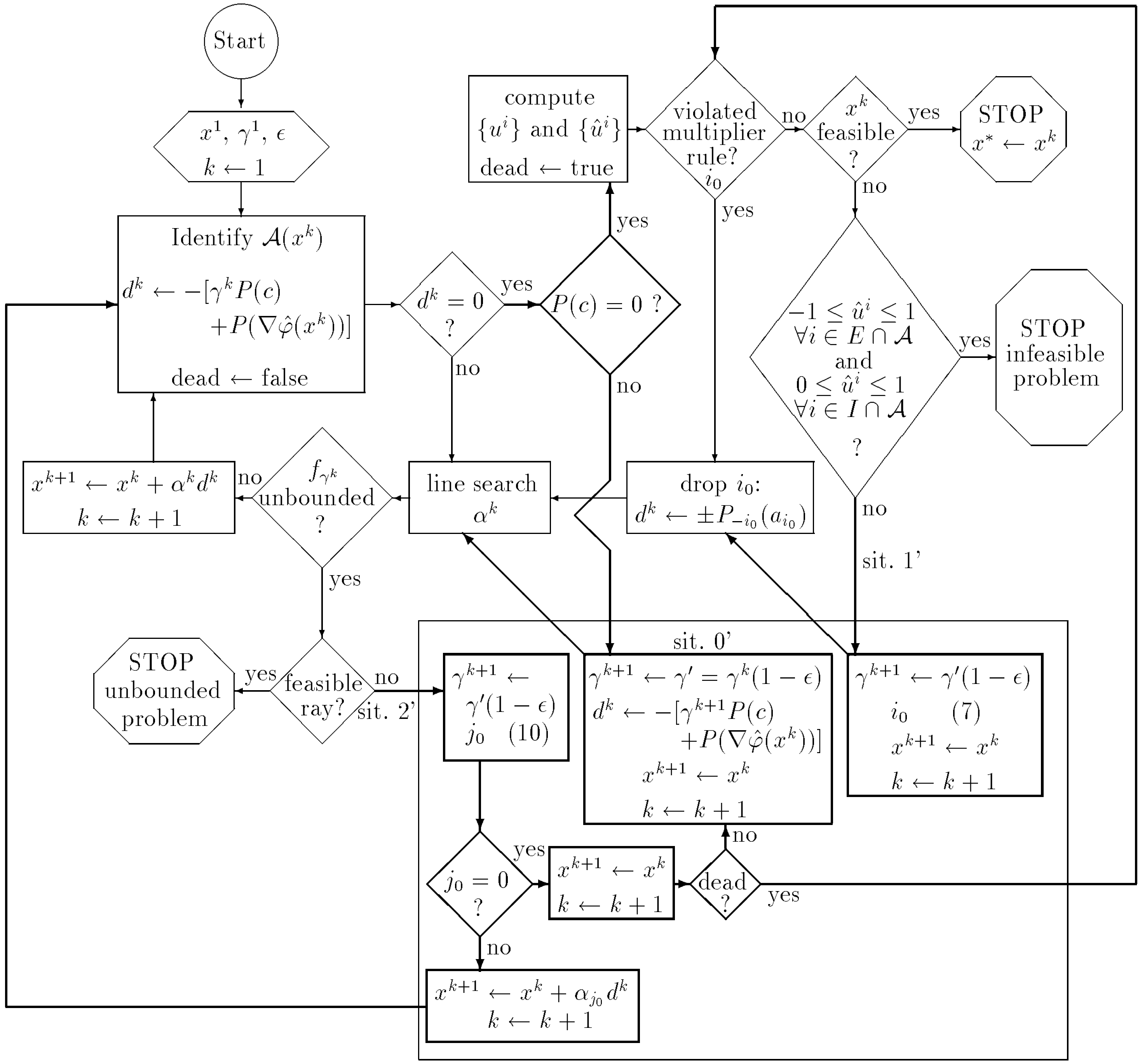

Figure 2: The new procedure for decreasing γ

4 Computational Experimentation

In order to investigate the effect of an automatic decrease of the penalty parameter, we have considered the minimum-cost network flow problem

$$\begin{array}{lc} \min_{x \in I\!R^n} & c^T x \\ \text{subject to} & Ax = b \\ & 0 \le x \le u, \end{array} \tag{11}$$

where A is the vertex-edge incidence matrix of a directed graph G, c is a vector of edge costs, u is a vector of edge capacities and b is a vector of vertex demands. We assume, without loss of generality, that G is connected.

Exterior penalty function methods originally developed for nonlinear programming were first specialized by Conn [6] and Bartels [1] to algorithms for solving linear programs. Gamble, Conn and Pulleyblank [12] then adapted these algorithms to a combinatorial one for the minimum-cost network flow problem (11). The basic idea of [12] is to include penalties in the objective function only for violated upper and lower bound constraints while explicitly imposing the equality constraints $Ax = b$, and to exploit the special structure present in these equality constraints. They thus consider the following penalty function subproblem

$$\begin{array}{lc} \min_{x \in I\!R^n} & f_\gamma(x) \\ \text{subject to} & Ax = b, \end{array} \tag{12}$$

where the penalty function is defined as

$$f_\gamma(x) = \gamma c^T x - \sum_{j=1}^{n} \min(0, x_j) - \sum_{j=1}^{n} \min(0, u_j - x_j). \tag{13}$$

In order to make comparisons, we have implemented three methods:

- The first one is the network simplex method. We have implemented a routine which is a particular implementation of the simplex algorithm specialized to network problems, along the lines described in [4], [13] and [14]. This routine solves problem (11) using *two* phases, one to find a feasible starting point and one to find the solution.

- The second method is the network penalty method of [12]. We have implemented the *network penalty algorithm* described in [12] to solve the equivalent problem to problem (11)

  $$\begin{array}{lc} \min_{x \in I\!R^n} & c^T x \\ \text{subject to} & Ax + Is = b \\ & 0 \le x \le u \\ & 0 \le s \le 0, \end{array}$$

  where I denotes the identity matrix, using the immediate (infeasible) starting point $(x, s) = (0, b)$. The code is an easy conversion of the one written for the simplex method, using the same data structure. The initial value of the penalty parameter γ used in that method has been fixed to 1 for all the tests and is divided by 10 after each infeasible termination.

- The third method is similar to the second one, where the automatic decrease of the penalty parameter described in section 3 has been included. As for the "classic" network penalty method, we have chosen to set the initial penalty parameter value to 1. Moreover, the parameter ϵ that arises in the penalty algorithm of section 3 is set to the relative machine precision value $\epsilon_M \simeq 1.39 \times 10^{-17}$.

We should emphasize here that the three codes that we have implemented could not be considered as "fast" network problem solvers. Our intention was simply to write network codes in a way that makes possible comparisons on the number of iterations required by each method.

We have tested the three methods on a set of ten randon test problems. These problems are the first ten of the set of thirty-five problems tested in [12] and generated by a revised version of NETGEN [15]. Table 1 reports the characteristics of these ten problems.

Problem number	Number of variables	Number of constraints
1	1308	200
2	1511	200
3	2000	200
4	2200	200
5	2900	200
6	3174	300
7	4519	300
8	5169	300
9	6075	300
10	6320	300

Table 1: Test problems

All the computations have been performed in double precision, on a DEC VAX 3500, under VMS, using the standard Fortran Compiler. The numerical results are reported in tables 2 and 3.

As observed in [12], table 2 shows that the percentage of improvement of the classic penalty method over the simplex method in terms of number of iterations is important. It also shows that using an automatic decrease of the penalty parameter can further improve the performance of the penalty method in terms of number of iterations. On the other hand, we observe in table 3 that the number of times the penalty parameter is decreased when using the automatic approach grows considerably. This phenomenon may be explained as follows. For each test, we have observed that the first time we had to decrease γ, we were in situation 1', that is, we were at an optimal solution for problem (12), but infeasible for problem (11). This means in our framework that no variable to enter the basis could be selected because no dual violation, as measured by the reduced cost, occurred. To force such a violation, we retain, in order to update γ, the *maximal* value such that at least one reduced cost generates a dual violation. But this implies such a small change in the reduced cost that the objectice function decreases very slowly and we do not go farther than the first bound encountered. That is, during the line search, the method behaves like the simplex method. As this update of γ also implies a very slow decrease of the penalty parameter value, this one needs to be decreased almost at each iteration and the above situation therefore occurs repeatedly. On the other hand, when comparing, in table 3, the last value of γ produced by both the classic and the automatic penalty methods, we observe very insignificant differences. As the improvement in terms of number of iterations (see table 2) is non-negligible, this confirms that it may be profitable not to decrease the penalty parameter too rapidly and to allow thereby infeasible steps to be taken during the optimization process.

Problem number	Simplex method	Penalty method ($\times 1/10$)	Percentage of improvement	Penalty method ($\times 1/10$)	Penalty method (automatic)	Percentage of further improvement
1	806	343	57.4	343	318	7.3
2	863	353	59.1	353	300	15.0
3	892	360	59.6	360	335	6.9
4	890	347	61.0	347	323	6.9
5	925	399	56.9	399	387	3.0
6	1354	597	55.9	597	539	9.7
7	1374	584	57.5	584	546	6.5
8	1559	617	60.4	617	537	13.0
9	1502	618	58.9	618	526	14.9
10	1696	693	59.1	693	605	12.7

Table 2: Number of iterations

Problem number	Number of decreases of γ		Last value of γ ($\times 10^{-3}$)	
	$\times 1/10$	automatic	$\times 1/10$	automatic
1	4	318	0.1	0.16
2	4	299	0.1	0.17
3	4	335	0.1	0.18
4	4	322	0.1	0.15
5	4	386	0.1	0.20
6	4	536	0.1	0.15
7	4	545	0.1	0.18
8	4	531	0.1	0.17
9	4	520	0.1	0.20
10	4	603	0.1	0.18

Table 3: Behaviour of γ

We have observed similar cpu times for the simplex method and the classic penalty method. Cpu times for the automatic penalty method were about twice higher. Note however that since the implementation of the automatic decrease of the penalty parameter requires to keep explicit the dependence of several quantities, as the reduced costs for instance, upon the penalty parameter γ, it doubles the amount of computation. Indeed, we must perform most of the computations *both* on the penalty part and the objective part of each quantity. Our implementation did not attempt to optimize this extra work.

The numerical results reported in tables 2 and 3 were obtained using a *total pricing* for the three methods. By total pricing, we mean that the entering variable is chosen as the one with the largest dual violation (we refer the reader to [12] for more details). The numerical results displayed in [12] are also based on total-pricing implementations. The authors in [12] yet noticed that it would be interesting to study the impact of the use of a *partial pricing*. In order to address this issue, we have also implemented the three methods using such a pricing. More precisely, rather than choosing the entering variable as the one with the most negative reduced cost among *all* the nonbasic variables, we select sets of thirty variables taken at regular intervals among the nonbasic variables and test each variable in the successive sets until a candidate to enter the basis is found. As mentioned in [12], partial pricing generally increases the number of iterations required but reduces execution times, which is of particular interest when solving large-scale problems. The results of table 4 effectively show this increase in the number of iterations, at least for the simplex and classic penalty methods. However, the number of iterations required by the automatic penalty method remains quite similar. This is not surprising since, at almost each iteration, the penalty parameter is decreased by such a small amount that no more than one nonbasic variable may be a candidate to enter the basis. We remark moreover that the percentage of improvement of the classic penalty method over the simplex method in terms of number of iterations is less important when using a partial pricing than a total pricing. On the other hand, this percentage clearly increases in favour of the automatic penalty method. The execution times observed are this time in favour of the simplex method, but once again this must be interpreted with great care.

Problem number	Simplex method	Penalty method ($\times 1/10$)	Percentage of improvement	Penalty method ($\times 1/10$)	Penalty method (automatic)	Percentage of further improvement
1	1065	595	44.1	595	318	46.6
2	1178	744	36.8	744	300	59.7
3	1297	788	39.2	788	336	57.4
4	1270	888	30.1	888	325	63.4
5	1535	1401	8.7	1401	386	72.4
6	2213	1336	39.6	1336	539	59.7
7	2453	1663	32.2	1663	547	67.1
8	2664	1854	30.4	1854	539	70.9
9	2851	1821	36.1	1821	526	71.1
10	2843	2527	11.1	2527	604	76.1

Table 4: Number of iterations, using partial pricing

5 Conclusions

We described an alternative way to decrease the penalty parameter in exact penalty methods based on the idea of performing the unconstrained minimization computations on both the objective part and the penalty part of f_γ. The theoretical idea, which aimed at avoiding decreasing the penalty parameter γ more than necessary in order to leave the situation which required a decrease of γ, is attractive since, in an algorithm making use of an updated factorization of the matrix of activities, computing two projections, both using the same factorization, rather than one should not represent significantly more work. The same comment applies to the two least-squares problems which have to be solved, as they differ only by their right-hand sides. Further investigations should address the issue of exploiting such features in order to optimize the cpu time required by the implementation of our method.

Computational experimentation on minimum-cost network flow problems reveals improvement on the number of iterations required to reach optimality in comparison with the penalty method of [12], which itself outperformed the network simplex method in terms of number of iterations. Our numerical experiments also show how important may be the choice of the penalty parameter, its impact on the number of iterations, and the viability of non-heuristic choices based on theoretical considerations. Moreover, the parametric representation of f_γ, φ_γ and $\{u^i_\gamma\}_{i\in\mathcal{A}}$, spares us most of the calculations in the step following one in which we decrease γ. We stress the fact that this applies also to the more standard method of decreasing γ—dividing it by a constant—if it integrates this idea of parametrizing γ. Finally, the idea should be extendable to other exact penalty functions. The fact that we developed the method based on an active-set approach algorithm makes the idea also generalizable to piecewise-linear programming, following the lines of [8], and to nonlinear constraints. Further work could attempt to address moreover second-order methods on nonlinear objective functions. This will however not be a straightforward task, as the parametrization of the penalty parameter implies an explicit dependence of the Hessian upon γ.

Acknowledgement

The authors are indebted to Andrew R. Conn for interesting discussions on the subject and also for his useful suggestions and comments on this paper.

References

[1] R. H. Bartels. A penalty linear programming method using reduced-gradient basis-exchange techniques. *Linear Algebra and its Applications*, 29:17–32, 1980.

[2] R. H. Bartels, A. R. Conn, and Y. Li. Primal methods are better than dual methods for solving overdetermined linear systems in the l_∞ sense? *SIAM Journal on Numerical Analysis*, 26:693–726, 1989.

[3] R. H. Bartels, A. R. Conn, and J. W. Sinclair. Minimization techniques for piecewise differentiable functions: the l_1 solution to an overdetermined linear system. *SIAM Journal on Numerical Analysis*, 15:224–241, 1978.

[4] G. H. Bradley, G. G. Brown, and G. W. Graves. Design and implementation of large scale transhipment algorithms. *Management Science*, 24:1–34, 1977.

[5] T. F. Coleman and A. R. Conn. Second-order conditions for an exact penalty function. *Mathematical Programming*, 19:178–185, 1980.

[6] A. R. Conn. Linear programming via a nondifferentiable penalty function. *SIAM Journal on Numerical Analysis*, 13:145–154, 1976.

[7] A. R. Conn and M. Mongeau. Discontinuous piecewise differentiable optimization I: Theory. Technical Report INRIA No1694, Institut National de Recherche en Informatique et en Automatique, Rocquencourt, BP 105, 78153 Le Chesnay, France, 1992.

[8] A. R. Conn and M. Mongeau. Discontinuous piecewise differentiable optimization II: Degeneracy and applications. Technical Report CRM-1869, Centre de recherches mathématiques, Université de Montréal, CP 6128-A, H3C 3J7 Canada, 1993.

[9] G. Di Pillo and L. Grippo. Exact penalty functions in constrained optimization. *SIAM Journal on Control and Optimization*, 27(6):1333–1360, 1989.

[10] A. V. Fiacco and G. P. McCormick. *Nonlinear Programming: Sequential Unconstrained Minimization Techniques.* John Wiley, 1968. (Reprinted SIAM, 1990).

[11] R. Fletcher. *Practical Methods of Optimization.* Wiley-Interscience, second edition, 1987.

[12] A. B. Gamble, A. R. Conn, and W. R. Pulleyblank. A network penalty method. *Mathematical Programming*, 50:53–73, 1991.

[13] M. D. Grigoriadis. An efficient implementation of the network simplex method. *Mathematical Programming Studies*, 26:83–111, 1986.

[14] J. L. Kennington and R. V. Helgason. *Algorithms for Network Programming.* John Wiley, 1980.

[15] D. Klingman, A. Napier, and J. Stutz. NETGEN: A program for generating large-scale capacitated assignment, transportation, and minimum cost flow network problems. *Management Science*, 20:814–821, 1974.

[16] M. Mongeau. *Discontinuous Piecewise Linear Optimization.* PhD thesis, Dept. of Combinatorics & Optimization, University of Waterloo, Ontario, Canada, 1991.

[17] A. Sartenaer. *On Some Strategies for Handling Constraints in Nonlinear Optimization.* PhD thesis, Department of Mathematics, Facultés Universitaires Notre-Dame de la Paix, Namur, Belgium, 1991.