Olga Karelkina | Yury Nikulin | Marko M. Mäkelä

An adaptation of NSGA-II to the stability radius calculation for shortest path problem

Turku Centre *for* Computer Science

TUCS Technical Report
No 1017, September 2011

An adaptation of NSGA-II to the stability radius calculation for shortest path problem

Olga Karelkina
University of Turku, Department of Mathematics
FI-20014 Turku, Finland
volkar@utu.fi
Yury Nikulin
University of Turku, Department of Mathematics
FI-20014 Turku, Finland
yurnik@utu.fi
Marko M. Mäkelä
University of Turku, Department of Mathematics
FI-20014 Turku, Finland
makela@utu.fi

TUCS Technical Report
No 1017, September 2011

Abstract

Abstract: This paper addresses two different approaches to the calculation of stability radius of an optimal solution to the well-known shortest path problem. We present an adaptation of multi-objective evolutionary algorithm (NSGA-II) to the considered problem. We also compare behavior of the derived algorithm with the known exact method in terms of solutions diversity and computational complexity. Algorithmic performance is tested by numerical experiments.

Keywords: combinatorial optimization; stability analysis; shortest path problem; stability radius; Pareto set: genetic algorithm.

1 Introduction

As a rule while solving various applied problems, the information for mathematical models construction is given inaccurately. This inaccuracy is caused by the various factors of uncertainty and randomness such as inadequacy of mathematical models to real processes, rounding off, calculation errors and etc. In these cases a mathematical problem cannot be properly solved without using results of stability theory (at least implicitly).

In this work we address the issue of deriving algorithm for calculation of quantitative characteristic of solution stability for the well known shortest path problem. A quantitative characteristic called stability radius is defined as the limit level of perturbations of problem parameters preserving optimality of a single solution (or of the solution set). The perturbed parameters are usually coefficients of the scalar or vector objective function.

So far, to the best of our knowledge, algorithms for calculating or estimating stability radii have only been created for some scalar problems. For example, Chakravarti and Wagelmans [2] developed an approach to constructing a polynomial algorithm for calculating the stability radius for some classes of polynomially solvable problems. However, the calculation time increases dramatically for large scale problems. Therefore, it is quite reasonable to apply modern heuristics in order to derive faster algorithms for the stability radius calculation. Moreover these methods can be further applied for calculation of stability radius of optimal solutions to various multicriteria discrete optimization problems.

Evolutionary algorithms (EAs) [3, 10, 16] are adaptive heuristic search algorithms based on the evolutionary ideas of natural selection and genetics. As such they represent an intelligent exploitation of a random search used to solve optimization problems. Although randomized they utilize historical information to direct the search into the region of better performance within the search space. At each generation, a new set of approximations is created by the process of selecting individuals according to their level of fitness in the problem domain and breeding them together using the evolution of populations of individuals that are better suited to their environment than their ancestors, just as in natural selection.

In this paper, non-dominated sorting genetic algorithm (NSGA-II) [3] based approach is proposed for calculating stability radius of an optimal solution to the single criterion shortest path problem. The essentially new idea we introduce here is to split the problem into the bi-criteria optimization problem of finding the Pareto set in order to concern fractional term minimization appearing in the analytical expression for the stability radius. The preference is given to NSGA-II instead of evolutionary algorithm for single criterion case because NSGA-II is a multi agent method providing a diversity among non-dominated solutions by using the crowding comparison

procedure. This procedure is used in the tournament selection and during population reduction phase. Moreover NSGA-II performs more sophisticated search than a straightforward genetic algorithm.

2 Basic definitions and notations

Given a directed graph $G = (V, A)$, where V is a set of vertices and A is the set of edges with cardinality $|V| = m$ and $|A| = n$. Each edge $e_i \in A$ is associated with positive cost c_i, $i \in N_n = \{1, 2, \ldots, n\}$. The shortest-path problem (SP) is the problem of finding a directed path from a distinguished source node s to a distinguished terminal node t, with the minimum total cost.

We can formulate SP as a linear programming problem by first defining the cost vector $C = (c_1, c_2, \ldots, c_n) \in \mathbf{R}^n_+$, $c_i > 0$. Denote by $X \subseteq 2^{\mathbf{E}^n}$, $\mathbf{E} = \{0, 1\}$, the set of feasible solutions, i.e. the set of all valid directed paths $P = (e_{i_1}, e_{i_2}, \ldots, e_{i_k})$ from node s to node t. Then the decision variables specify all possible paths in graph G:

$$x_i = \begin{cases} 1, & \text{if } e_i \in P, \\ 0 & \text{otherwise.} \end{cases} \tag{1}$$

Let us define by $e_i \to j$ an edge for which vertex i is a tail and vertex j is a head and by $e_i \leftarrow j$ an edge for which vertex j is a tail and vertex i is a head. Then boolean linear programming formulation is given as follows:

$$\sum_{e_i \in A} c_i x_i \to \min, \tag{2}$$

subject to

$$\sum_{e_i : e_i \to j} x_i - \sum_{e_i : e_i \leftarrow j} x_i = \begin{cases} 1, & \text{if } j = s, \\ -1, & \text{if } j = t, \\ 0 & \text{otherwise.} \end{cases} \tag{3}$$

$$x_i \in \{0, 1\}. \tag{4}$$

Here (3) is classical network flow balance constraints and (4) is Boolearity constraints which define the set of feasible solutions (paths).

It is well known that if we omit the requirement of variables boolearity assuming $x_i \in [0, 1]$, $i = (1, 2, \ldots, n)$, the relaxed linear programming problem will have an integer (boolean) solution due to the total unimodularity of the constraints matrix [14]. Thus, in principle, the shortest path problem can be solved as a linear programming problem. However, being more efficient problem oriented discrete algorithms are used in practice. The considered problem with positive costs can be easily solved by Dijkstra's algorithm [4], which processes nodes in nondecreasing order of their actual

distances from the source node. At the beginning all nodes are given an infinite distance except the source which is given a distance 0. At each step we choose the next unlabeled node which is nearest to the source and mark it, while updating the optimal distance to all its neighbors. The optimal distance of a neighbor is updated only if reaching it from the current labeling node gives a total path length that is shorter then its current distance. Doing so the algorithm constructs the so-called shortest path tree, which is a spanning tree rooted at the source node s where the shortest paths to all other nodes are determined. The shortest path to each node is then found by tracing the predecessor iteratively back to the source. One of the best implementations of Dijkstra's algorithm uses priority queue structure [6] and has time complexity $O(m\ log(m+n))$ (see, e.g. [1]). Observe that Dijkstra's algorithm can be correctly applied to the problem only in the case when all $c_i > 0,\ i = (1, 2, \ldots, n)$, otherwise it terminates but does not provide correctness, i.e. proper optimal solution is unlikely to be found. In the case with negative costs and at the presence of negative costs cycles the algorithm experiences also a problem with termination.

We define norms l_1 and l_∞ in $\mathbf{R}^d$ for any finite dimension $d \in \mathbf{N}$:

$$\|y\|_1 = \sum_{i \in N_d} |y_i|, \qquad \|y\|_\infty = \max\{|y_i| \mid i \in N_d\},$$

where $y = (y_1, y_2, \ldots, y_d)^T \in \mathbf{R}^d$ and $\mathbf{N}_d = \{1, 2, \ldots, d\}$.

The perturbation of the problem parameters is understood as an arbitrary independent change of coefficients of objective function (2). A perturbation is modeled by adding to vector C a perturbing vector $C' = (c'_1, c'_2, \ldots, c'_n)$ from the set

$$\Omega(\varepsilon) = \{C' \in \mathbf{R}^n \mid \|C'\|_\infty < \varepsilon\},$$

where $0 < \varepsilon < \min\{c_i \mid i \in N_n\}$. Thus, we always preserve that $c_i + c'_i \geq 0$ for any $i \in N_n$.

Limiting perturbations we consider only small changes of problem parameters. One should mention that this restriction is not unusual because for problems occurring in practice typically all data are given with some kind of accuracy ratio which could not exceed the nominal value.

The perturbed problem is formulated as follows:

$$\sum_{e_i \in A} (c_i + c'_i) x_i \to \min, \tag{5}$$

subject to (3) and (4).

In this situation, it makes sense to estimate a quantitative characteristic of an optimal solution stability. Such a characteristic called stability radius is defined as the limit level of perturbations for which a certain relation between solutions of problems (2)–(4) and (5), (3), (4) holds. If the level of

uncertainty in problem parameters is not greater than the stability radius, then the solution of (2)–(4) is practically relevant (in certain sense).

Denote by $X_{opt}(C)$ the set of optimal solutions to the initial problem (2)–(4) with cost vector C.

Definition 1. An optimal solution $x \in X_{opt}(C)$ is called *stable* if there exists $\varepsilon > 0$ such that for any $C' \in \Omega(\varepsilon)$ we have $x \in X_{opt}(C + C')$.

Definition 2. *Stability radius* of $x \in X_{opt}(C)$ formally can be defined as follows

$$\rho(x, C) = \begin{cases} \sup \Theta, & \text{if } \Theta \neq \emptyset, \\ 0, & \text{if } \Theta = \emptyset, \end{cases}$$

where

$$\Theta = \{\varepsilon > 0 \mid \forall C' \in \Omega(\varepsilon),\ x \in X_{opt}(C + C')\}.$$

In other words, the stability radius of x is the supremum level of parameter perturbations such that x remains optimal in the perturbed problems under given restricted perturbations. If x remains optimal for any arbitrary large perturbations, then its stability radius is assumed to be infinite.

The formula for stability radius of an optimal solution to the single criterion linear programming problem was originally obtained in [13] and expressed as follows:

$$\rho(x, C) = \min_{x' \in X \setminus \{x\}} \frac{\sum\limits_{i \in \mathbf{N}_n} c_i(x'_i - x_i)}{\|x' - x\|_1}. \tag{6}$$

Under our assumption about "small perturbations" for the considered problem this formula transforms into

$$\rho(x, C) = \min \left\{ \min_{i \in N_n} c_i, \min_{x' \in X \setminus \{x\}} \frac{\sum\limits_{i \in \mathbf{N}_n} c_i(x'_i - x_i)}{\|x' - x\|_1} \right\}. \tag{7}$$

In further research, formula (6) was generalized on various classes of vector discrete optimization problems (see, e.g. [7, 8]).

3 Exact method of calculating stability radius

In the case of a single objective function, Chakravarti and Wagelmans [2] presented an approach to calculating the stability radius of an ε-solution to the linear problem of 0–1 programming in polynomial time. They assumed that the objective function is minimized, the feasible solution set is fixed and a given subset of the objective function coefficients is perturbed. The approach requires that the original problem is polynomially solvable.

The method of Chakravarti and Wagelmans relies on the following theorem.

Theorem 1 *[2] Let x be an optimal solution to (2)–(4). The stability radius $\rho(x, C)$ of x is the maximum ρ satisfying the following inequality*

$$\min_{x' \in X \setminus \{x\}} \sum_{i \in N_n} (c_i - \rho d_i) x' \geq \sum_{i \in N_n} (c_i + \rho) x_i, \tag{8}$$

where

$$d_i = \begin{cases} 1, & \text{if } x_i = 0, \\ -1, & \text{if } x_i = 1. \end{cases}$$

In fact, in [2], a more general statement is formulated, but we restrict it to the context of the shortest path problem.

It is shown in [9] that Theorem 1 can be obtained from the formula (6) of stability radius. Thus having a formula of stability radius for a single criterion problem, one can derive an inequality analogous to (8).

The right-hand side of (8) is a linear function of ρ. The left-hand side is the value function of a parametric version of problem (2)–(4), where the objective coefficients are linear functions of ρ. Let us call this value function $v(\rho)$. It is well known (see, for instance, [5] or [11]) that $v(\rho)$ is a continuous, piecewise linear and concave function of ρ. It follows from the results of [2] that the number of linear pieces of $v(\rho)$ is $O(m)$, where m is number of vertices in graph G.

In [11] a general scheme for solving linear parametric computing problems was presented and was applied for constructing function $v(\rho)$ on $[0, \rho_u]$ in [2], where ρ_u is an upper bound for the value of stability radius and theoretically equals $\|C\|_\infty$ (see, for instance, [9]), but in our case, as was mentioned earlier, $\rho_u = \min_{i \in N_n} c_i$. The method starts with computing $v(0)$ and $v(\rho_u)$. The optimal solutions associated with these values each defines a linear function on $[0, \rho_u]$. If these linear functions are identical, then $v(\rho)$ is simply this linear function. Otherwise, we have two linear functions which intersect at a unique value $\bar{\rho} \in [0, \rho_u]$. If $(\bar{\rho}, v(\bar{\rho}))$ coincides with the intersection point, then $v(\rho)$ is the concave lower envelope of the two linear functions. Otherwise, the optimal solution associated with $\bar{\rho}$ defines a third linear function which intersects each of the other linear functions on $[0, \rho_u]$. These two intersection points define new values of ρ for which $v(\rho)$ is to be computed, and so on.

Once $v(\rho)$ has been computed, it is trivial to find the largest value of ρ for which this function is greater than or equal to the linear function $\sum_{i \in N_n} (c_i + \rho) x_i$.

The running time of the described procedure is shown in [5] to be $O(BR(m))$, where B is the number of linear pieces of $v(\rho)$ and $R(m)$ is the complexity of solving an instance of (2)–(4), i.e. the complexity of Dijkstra's algorithm that solves the single-source shortest path problem for a graph with nonnegative edge path costs. In worst case we need to analyze $O(m^2)$ linear pieces

(or breakpoints) for constructing function $v(\rho)$. Thus, the total complexity of the exact method is $O(m^4)$.

4 NSGA-II adaptation for calculating stability radius

A genetic algorithm (GA) is a search and optimization method which works by mimicking the evolutionary principles and chromosomal processing in natural genetics. Originally GA technique was proposed for single criterion case and was further extended on finding Pareto optimal solutions to multiobjective optimization problems. A GA begins its search with a random set of solutions usually coded in binary string structures. Every solution is assigned a fitness which is directly related to the objective functions of the search and optimization problem. Thereafter, the population of solutions is modified to a new population by applying three operators similar to natural genetic operators – reproduction, crossover, and mutation. A GA works iteratively by successively applying these three operators in each generation till a termination criterion is satisfied. Similar to natural selection each simulation run is aimed to improve new generation in the sense that a fitness value is optimized. Over the past one decade and more, GAs have been successfully applied to a wide variety of problems, because of their simplicity, global perspective, and inherent parallel processing.

In order to apply a non-dominated sorting based multi-objective evolutionary algorithm (or NSGA-II) proposed in [3] to calculation of the stability radius we suggest to treat the minimization of fraction (6) as bi-objective discrete optimization problem:

$$f_1 := \sum_{i \in \mathbf{N}_n} c_i(x'_i - x_i) \to \min_{x' \in X \setminus \{x\}}$$

$$f_2 := \|x' - x\|_1 \to \max_{x' \in X \setminus \{x\}}$$

The Pareto set of the problem is formally defined as follows:

$$P^2(C) = \{x' \in X \mid \nexists x \in X \ \big(f_1(x, C) \le f_1(x', C) \wedge f_2(x, C) \ge f_2(x', C)\big) \wedge$$

$$\wedge \big(f_1(x, C) \ne f_1(x', C) \vee f_2(x, C) \ne f_2(x', C)\big)\}. \qquad (9)$$

In other words, a feasible solution is Pareto efficient if there is no feasible solution which strictly dominates by one of the objectives and not worse by any other. Here the first objective function is numerator of fraction (6) which should be minimized and the second objective function is denominator of fraction (6) which should be maximized in order to obtain the minimum value of fractional ratio (6). Thus it is evident that the value of stability

radius corresponds to one of the points from Pareto frontier which delivers minimum to the fraction.

In the following, we discuss the details of the coding representation of a solution and present developed operators of NSGA-II adaptation for stability radius calculation. The proposed algorithm uses a fast non-dominated procedure and an elitist-preserving approach [3].

4.1 Representation and initialization

We consider a randomly generated directed graph G. The topology of the graph is defined by an adjacency matrix and costs of edges are defined by a cost matrix. A chromosome or a solution consists of integer numbers of nodes that form a path from the source node to a terminal node. The length of the chromosome is variable and may not be greater than number of nodes m. Let us consider the example of digraph with the following adjacency and cost matrices

$$A_0 = \begin{pmatrix} 0 & 1 & 1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix}$$

$$C_0 = \begin{pmatrix} \infty & 30 & 43 & \infty & \infty & \infty & 9 & \infty & 19 & \infty \\ \infty & \infty & 15 & 4 & 17 & \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & 21 & 1 & \infty & \infty & \infty & \infty & \infty \\ \infty & \infty & \infty & \infty & \infty & 26 & \infty & \infty & 30 & \infty \\ 44 & \infty & \infty & \infty & \infty & \infty & 40 & \infty & \infty & \infty \\ \infty & 43 & \infty & \infty & 34 & \infty & \infty & 5 & \infty & 15 \\ 45 & 28 & \infty & \infty & 28 & 31 & \infty & \infty & 29 & \infty \\ \infty & \infty & \infty & \infty & \infty & \infty & 36 & \infty & \infty & 28 \\ \infty & 5 & 33 & \infty & \infty & \infty & \infty & \infty & \infty & 22 \\ \infty & \infty & \infty & \infty & \infty & \infty & \infty & \infty & 49 & \infty \end{pmatrix}.$$

The image of the graph is given in figure 1. The representation of one path from node 1 to node 10 is either $(1, 3, 4, 6, 10)$ or $((1, 3), (3, 4), (4, 6), (6, 10))$.

Initial population of solutions which represent valid directed paths from source node s to terminal node t is generated using breadth first search algorithm [15]. This method is chosen in order to guarantee diversity of

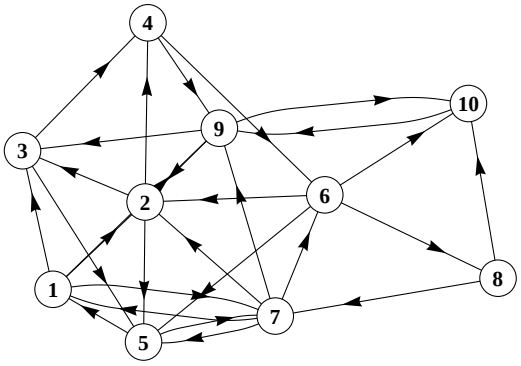

Figure 1: A random directed graph on 10 nodes.

solutions in the randomly generated initial population. In current implementation of NSGA-II population size is equal to the number of nodes in the considered graph.

It is necessary to note that we create an initial population only based on graph topology without deriving information about the cost vector. One can certainly produce a more efficient starting population by applying Dijkstra's algorithm to solve the problem of type (8) (left hand side) for several randomly picked up values of ρ. However it will compromise the idea to avoid the exact algorithm within the heuristic. This will have greater importance, if we would like to generalize the approach for some NP hard problem.

4.2 A fast non-dominated sorting approach

After a random parent population is created it is sorted based on the non-domination level using a fast non-dominated sorting procedure [3]. In this approach, every solution from the population is checked with a partially filled population for domination. When all solutions of the population are checked, the first non-dominated class for all solutions is found. To find other fronts, the members of the first class are discounted from population and sorting procedure is repeated. At the end of the operation, all solutions are sorted and each front is stored separately.

This method requires a maximum of $O(K^2)$ domination checks, where K is the number of generations and in our case equals the number of nodes m. Since each domination check requires two function value comparisons, the maximum complexity of this approach to find the first non-dominated front is $O(2m^2)$.

One thing worth mentioning is that since elitism is introduced by compar-

ing current population with previously-found best non-dominated solutions, the procedure is different after the initial generation.

4.3 Reproduction

Reproduction (or selection) operator is applied on a population directly after sorting procedure. Selection plays an important role in improving the average quality of the population by passing the high quality chromosomes to the next generation. Selection process is guided by the crowded comparison operator [3]. Let us assume that every individual in the population has two attributes: non-domination rank that is front number and crowding distance which serves as an estimate of the size of the largest cuboid enclosing the point without including any other point in the population. That is, between two solutions with differing non-domination ranks we prefer the solution with lower (better) rank. Otherwise, if both solutions belong to the same front then we prefer the solution which is located in a less crowded region.

4.4 Crossover operators

Crossover operators are applied next to the chromosomes selected by reproduction procedure. For current implementation of NSGA-II we use three different types of crossovers, so called one-node (ON), one-edge (OE) and one-node-two-edges (ONTE) crossovers. At first solutions which correspond to the minimum and the second minimum value of fraction (6) are picked up from the current generation.

ON crossover is an analog of the conventional one-point crossover [12] with one difference is that two chromosomes chosen for crossover should have at least one common gene (node) except for source and destination nodes. ON operator builds an offspring by choosing one random node among common nodes for two paths and swapping substrings of two paths before or after cut point. One thing worth mentioning is that all crossover operators used in the proposed algorithm check if there are common nodes in substrings from different paths before swapping and joining procedure in order two avoid appearance of cycles and produce only feasible solutions. For the graph in figure 1 we can choose the following two paths as parents: $(1, 2, 5, 7, 6, 10)$ and $(1, 3, 5, 7, 9, 10)$ (see figure 2). Node 7 is common for both parents. Crossover exchanges substrings of the first and second parents from node 1 to node 7 producing two offsprings illustrated in figure 3. One-node crossover is very straightforward to implement. However, it can not be applied to any pair of paths, because they may not have a common vertex. Moreover, this crossover is not capable to produce offsprings with edges that do not belong to parents. Thus, it is worth to consider other types of crossover operators.

OE crossover works as follows. First, two random edges that connect two

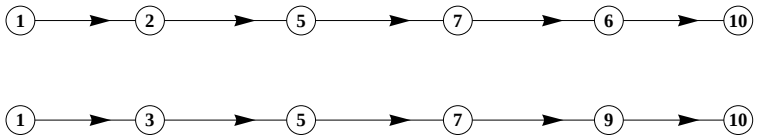

Figure 2: Two parents selected for crossover procedure

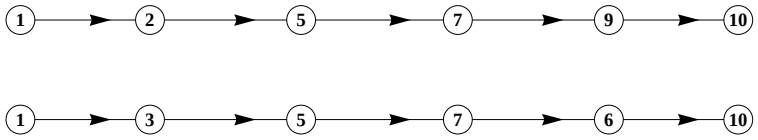

Figure 3: Two offsprings produced by one-node crossover

paths are selected. For instance, edges $(2, 3)$ and $(1, 7)$ can be selected for parents in figure 2. Then substring of the first parent from node 1 to node 2 is combined with edge $(2, 3)$ and substring of the second parent from node 3 to node 10. The same procedure is repeated for edge $(1, 7)$ and corresponding substrings. The obtained offsprings are displayed in figure 4.

Note that OE crossover allows to find paths with edges which do not belong to parents. Moreover, the probability of applying this crossover to two random chromosomes is higher than for ON crossover, especially in dense graphs. In addition, using OE technique we can produce more than two offsprings and as a result find more good candidates. However, it is not possible to build paths containing nodes which do not belong to the parents. Therefore, we introduce the third model of crossover.

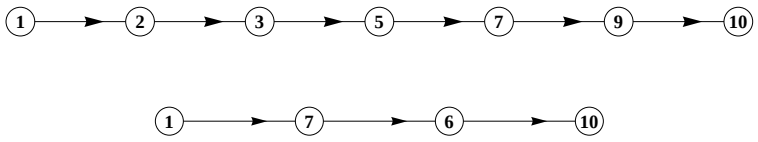

Figure 4: Two offsprings produced by one-edge crossover

ONTE crossover builds offspring by selecting random node which does not belong to parents and joining substring of one parent with a substring of

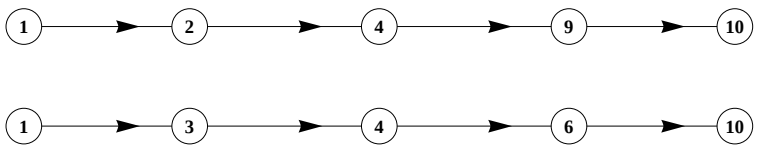

Figure 5: Two offsprings produced by one-node-two-edges crossover

the other parent using two edges adjacent to the selected node. For example two paths in figure 2 would produce offsprings as follows. First, a random connecting node that is not in paths is chosen, e.g. node 4. Next, crossover founds all possible edges whose tail is adjacent to vertices of the first path and head is adjacent to the connecting node, e.g. edge $(2, 4)$, and all possible edges whose tail is adjacent to the connecting node and head is adjacent to the second path, e.g. edge $(4, 9)$. The same procedure is repeated in the other direction, from the second path to the first one. Thus, two edges links $((2, 4), (4, 9))$ and $((3, 4), (4, 6))$ are formed that connect substrings of the first (second) path with substrings of the second (first) path. Finally, substring of the first (second) path from node 1 to the node adjacent to link is glued with link itself and substring of the second (first) path from the node adjacent to the link to the terminal node 10 (see figure 5).

Observe that ONTE crossover applied to two chromosomes can produce quite many offsprings but only two of them are selected to the new generation. In order to pick up the best two strings, all created solutions are sorted according to the value of the ratio of two functions, f_1 and f_2, and those which have minimal value of this ratio are included into the new generation. This procedure helps to direct the local search towards the optimal solution, but can slow down run time of the algorithm. Therefore, combining of three different crossover operators leads to more efficient evolutionary algorithm.

4.5 Mutation

The search of genetic algorithm is mainly guided by crossover operators, even though mutation is also used to maintain diversity in the population. Furthermore, mutation is useful for local improvement of a solution. Here we suggest two types of mutations. One takes a path of length greater then 2 and randomly selects a subpaths which consists of two edges and replace it by one edge if it is possible. Another type of mutation randomly selects two adjacent nodes in the path and replace an edge that connect them by two edges link if such exists in the considered graph. For example, in the graph in figure 1 path $(1, 9, 10)$ can be obtained from path $(1, 7, 9, 10)$ by applying first

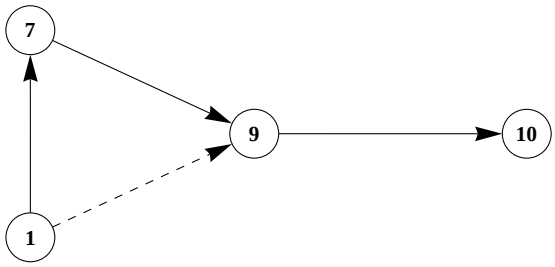

Figure 6: Mutations scheme

type of mutation and visa versa by applying second type. In the proposed algorithm probability of applying mutation to some chromosome increases with the number of population and probability of what type of mutation to choose is one half.

5 Comparison of the exact method and NSGA-II

Let us now look at the time complexity of one iteration of the entire NSGA-II. As it was shown in [3] the overall complexity of the above algorithm is $O(Km^2)$, where K is the number of generations, and it is governed by the non-dominated sorting part. In other words, in our case it is $O(10m^2)$ or $O(20m^2)$, while the exact method complexity is $O(m^4)$.

To test accuracy of our algorithm, we use randomly generated directed graphs on 100 nodes with approximately 5000 edges. The population size is set to 100 for all tests. The ONTE crossover, the OE crossover and the ON crossover probabilities are 0.5, 0.3 and 0.2 correspondingly. Mutation probability in the first generation is 0.05 and for subsequent generations it is calculated by the formula $0.05 + 0.45K_{current}/K_{total}$, where $K_{current}$ is the number of a current generation and K_{total} is the total number of generations in the test. This relation between mutation probability and generation number is not conventional and was chosen based on preliminary test runs.

In figure 7 we show computational results for ten different instances of the shortest path problem. In order to estimate the affect of the number of generations on the obtained results we performed test runs of NSGA-II both for 10 and 20 generations. Results of our algorithm are compared with those of Chakravarti and Wagelmans exact method. Accuracy of NSGA-II is measured by absolute error, that is the difference between the value obtained by NSGA-II and the exact value, and absolute error normalized by

Test instance	1	2	3	4	5	6	7	8	9	10
Costs interval	[1,100]	[1,100]	[1,100]	[1,100]	[1,100]	[5,200]	[5,200]	[5,200]	[5,200]	[5,200]
minimum c_i	1	1	1	1	1	5	5	5	5	5
NSGA-II, 10	$\frac{9}{10}$	$\frac{3}{8}$	1	$\frac{1}{2}$	$\frac{1}{7}$	$\frac{17}{5}$	$\frac{1}{2}$	$\frac{1}{3}$	$\frac{3}{2}$	$\frac{24}{5}$
NSGA-II, 20	$\frac{5}{9}$	$\frac{3}{8}$	1	$\frac{1}{2}$	$\frac{1}{7}$	$\frac{5}{2}$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{3}{7}$	$\frac{13}{3}$
CW	0	0	0	$\frac{1}{2}$	$\frac{1}{7}$	$\frac{5}{2}$	$\frac{1}{2}$	$\frac{1}{4}$	$\frac{3}{7}$	$\frac{13}{3}$
Absolute error	$\frac{5}{9}$	$\frac{3}{8}$	1	0	0	0	0	0	0	0
Normalized absolute error	$\frac{5}{9}$	$\frac{3}{8}$	1	0	0	0	0	0	0	0

Figure 7: Comparison of NSGA-II for 10 generations, NSGA-II for 20 generations and CW (method proposed by Chakravarti and Wagelmans)

the length of interval $[0, \min_{i \in N_n} c_i]$. Relative error does not provide any meaningful information if input (exact value) is 0. For the first three instances of the problem NSGA-II did not converge to the exact value of stability radius. Therefore, in some cases it is worth to increase the number of generations in order to improve an average accuracy of the proposed algorithm. In figure 7 it is clearly seen how the number of generations affects the results for $1, 6, 8, 9$ and 10 problem instances.

6 Conclusions

This work is the first attempt to derive an alternative heuristic approach to the stability radius calculation. Shortest path problem was chosen for testing based on the fact that the method proposed by Chakravarti and Wagelmans runs in polynomial time if the optimization problem itself is polynomially solvable. Thus, we can estimate accuracy of results of our approach by comparison with those of the exact method. As it was shown above, theoretical time complexity of adapted NSGA-II is competitive with complexity of the algorithm proposed by Chakravarti and Wagelmans. Moreover NSGA-II complexity can be reduced by decreasing the number of generations, however this could affect accuracy of solutions.

The apparent computational efficiency of the proposed algorithm is explained by the following two facts: at the beginning of the solution process, breadth first search procedure provides diversity in the initial population and chosen size of the population, which is equal to the number of nodes, is enough to generate good solutions and, in addition, keeps memory and time; combining three different crossover operators lead to more exhaustive search allowing to find solutions faster.

Preliminary comparisons showed that the convergence rate of the adapted NSGA-II was good for almost all random scenarios of the shortest path problem that were tested, though the number of tested instances could have been larger. This study encourage us to believe that our approach has some

real potential. In addition, the exact algorithm is not polynomial for NP hard problems, while NSGA-II has still complexity of $O(Km^2)$ since it does not depend on complexity of the original problem. Further, our emphasis is on applying algorithm working on similar principles as adapted NSGA-II for calculating stability radius for NP hard combinatorial optimization problems, e.g. TSP, and multi-criteria combinatorial optimization problems.

This research was partially supported by Doctoral Programme in Systems Analysis, Decision Making and Risk Management and by Belarusian Republican Fund of the Fundamental Research (project F10M-183).

References

[1] R. K. Ahuja, K. Mehlhorn, J. Orlin and R. E. Tarjan, Faster algorithms for the shortest path problem, Journal of the ACM 37 (2) (1990), 213–223.

[2] N. Chakravarti and A. Wagelmans, Calculation of stability radius for combinatorial optimization, Operations Research Letters 23 (1) (1998), 1–7.

[3] K. Deb , A. Pratap , S. Agrawal and T. Meyarivan, A fast and elitist multi-objective genetic algorithm NSGA-II, Evolutionary Computation 6 (2) (2002), 182–197.

[4] E. W. Dijkstra, A note on two problems in connection with graphs, Numerische Mathematik 1 (1959), 269–271.

[5] M. J. Eisner and D. G. Severance, Mathematical techniques for efficient record segmentation in large shared databases, J. Assoc. Comput. Mach. 23 (1976), 619–635.

[6] P. van Emde Boas, R. Kaas, and E. Zijlstra, Design and implementation of an efficient priority queue, Mathematical Systems Theory 10 (1977), 99–127

[7] V. A. Emelichev, E. Girlich, Yu. V. Nikulin and D. P. Podkopaev, Stability and regularization of vector problem of integer linear programming, Optimization 51 (4) (2002), 645–676.

[8] V. A. Emelichev, V. N. Krichko and D. P. Podkopaev, On the radius of stability of a vector problem of linear Boolean programming, Discrete Math. Appl. 10 (2000), 103–108.

[9] V. A. Emelichev and D. P. Podkopaev, Quantitative stability analysis for vector problems of 0 – 1 programming, Discrete Optimization 7 (2010), 48–63.

[10] D. E. Goldberg, B. Korb and K. Deb, Messy genetic algorithms: Motivation, analysis, and first results, Complex Syst. 3 (1989), 93–530.

[11] D. Gusfield, Parametric combinatorial computing and a problem of program module distribution, J. Assoc. Comput. Mach. 30 (1983), 551–563.

[12] J. H. Holland, Adaptation in Natural and Artificial Systems, Cambridge, M.A.: The MIT Press (1992)

[13] V. K. Leont'ev, Stability in linear discrete problems, Probl. Kiber. 35 (1979), 169–184.

[14] C. H. Papadimitriou and K. Steiglitz, Combinatorial Optimization: Algorithms and Complexity, Mineola, N.Y.: Dover Publications (1998)

[15] S. Russel and P. Norvig, Artificial Intelligence: a modern approach (2nd Ed.), Upper Saddle River, N.J.: Prentice Hall (2002)

[16] E. Zitzler, Evolutionary algorithms for multiobjective optimization: Methods and applications, Doctoral dissertation ETH 13398, Swiss Federal Institute of Technology (ETH), Zurich, Switzerland (1999).

Turku
Centre *for*
Computer
Science

Lemminkäisenkatu 14 A, 20520 Turku, Finland | www.tucs.fi

University of Turku
- Department of Information Technology
- Department of Mathematics

Åbo Akademi University
- Department of Computer Science
- Institute for Advanced Management Systems Research

Turku School of Economics and Business Administration
- Institute of Information Systems Sciences

ISBN 978-952-12-2632-8
ISSN 1239-1891