A genetic algorithm for robust schedules in a just-in-time environment

Marc Sevaux
University of Valenciennes
CNRS, UMR 8530, LAMIH / Production Systems, France
e-mail: marc.sevaux@univ-valenciennes.fr

Kenneth Sörensen*
University of Antwerp
Middelheimlaan 1, B-2020 Antwerp, Belgium
e-mail: kenneth.sorensen@ua.ac.be

December 2002

Abstract

Computing a schedule for a single machine problem is often difficult for irregular criteria, but when the data are uncertain, the problem is much more complicated. In this paper, we modify a genetic algorithm to compute robust schedules when release dates are subject to small variations. Two types of robustness are distinguished: quality robustness or robustness in the objective function space and solution robustness or robustness in the solution space. We show that the modified genetic algorithm can find solutions that are robust with respect to both types of robustness. Moreover, the risk associated with a specific solution can be easily evaluated. The modified genetic algorithm is applied to a just-in-time scheduling problem, a common problem in many industries.

Key words: Quality robustness, solution robustness, single machine scheduling problem, weighted number of late jobs, genetic algorithm. MSC Mathematics Subject Classification code 90B99.

1 Introduction

Scheduling problems are generally computationally difficult problems and the problem described in this paper is no exception. The problem becomes

*Corresponding author

even more difficult when some of the data are stochastic. Genetic algorithms—and metaheuristics in general—have been found to provide good approximations of the optimal solution with a relatively small computational effort. Although the literature on applications of metaheuristics to scheduling problems is very large, the number of applications to problems with a stochastic component is rather small.

One solution to deal with stochastic problem data is to find solutions that are *robust*. We distinguish two types of robustness. *Quality robustness* is a property of a solution whose quality, measured by the objective function value, does not deviate much from optimality when small changes in the problem data occur. The second type of robustness is *solution robustness* and can be described as robustness in the solution space. Solution robustness occurs when a solution does not change much when small changes in the problem data occur, i.e. when the solution method is able to find a solution that is "close" (in the solution space, not the objective function space) to the solution currently used.

The object of the *single machine scheduling problem* considered in this paper is to determine the order in which a given set of *jobs* should be executed on a machine. Each job has a certain release date and cannot be scheduled before this date. Also, each job has a certain processing time and a due date, before which it should be finished. Jobs that are not finished before their due date are called *late*. The objective of the problem considered in this paper is to minimise the number of late jobs. A genetic algorithm for this problem was developed by Sevaux and Dauzère-Pérès [24].

When a problem has stochastic parameters, the objective function value itself becomes stochastic. The objective of a stochastic problem has several dimensions. Some decision makers prefer a solution that has a high quality *on average*, others might prefer one that hedges against the worst possible situation. However, most decision makers like to minimise the risk of ending up with a bad solution, without necessarily choosing an extremely conservative solution. In this paper, we modify the GA [24] to be able to deal with stochastic release dates and show that the GA is able to find solutions that are both quality robust and solution robust. The proposed approach can be labelled a *sampling* method, in that it uses sampling to estimate the robustness of a solution. We show that this approach is very flexible in that—although the GA is only used for problems with stochastic release dates—other types of stochastic problem data can be easily incorporated. Simultaneously, the risk preference of the decision maker can be modelled and a solution can be found that minimises the estimated number of late jobs and the risk that this solution entails. Solution robustness is obtained by measuring the distance (in the solution space) to a given base-line solution. This distance measure is based on the *edit distance*.

This paper is structured as follows. After the literature review, the single machine scheduling problem is formulated. In section 4, the concept

of robustness is discussed and we explain its importance in scheduling. Section 5 introduces a genetic algorithm for the deterministic single machine scheduling problem. In section 6, the GA is modified for the computation of schedules that are both quality robust and solution robust. Section 7 describes how the modified genetic algorithm can be used to find robust production sequences in a just-in-time environment. Section 8 reports on some computational results.

2 Literature review

Single machine scheduling problems have been successfully tackled using genetic algorithms. Genetic algorithms—pioneered by Holland [15] and Goldberg [11], as well as many others—use a Darwinian metaphor to find good solutions for difficult optimisation problems. A review of genetic algorithms can be found in Reeves [22, 23]. For a review of applications of genetic algorithms to scheduling problems we refer to Portmann [21].

This paper describes a simple GA that finds good solutions to the deterministic scheduling problem described above, the GA was developed by Sevaux and Dauzère-Pérès [24].

Although the literature on deterministic scheduling is several orders of magnitude larger, there is a significant body of work on scheduling under uncertainty. Two very recent literature surveys are Herroelen and Leus [14] and Davenport and Beck [10]. Herroelen and Leus [14] divide research on scheduling with uncertainty into six categories. In *reactive scheduling*, procedures are developed to re-optimise the schedule when disruptive events occur. *Stochastic scheduling* is the application of stochastic programming with recourse to scheduling problems. *GERT network scheduling* is used for problems with stochastic evolution structure and feedback. *Scheduling under fuzziness* recommends the use of fuzzy numbers for e.g. project duration instead of stochastic variables. *Proactive (robust) scheduling* attempts to develop a schedule of which the quality is relatively insensitive to a changing environment. Herroelen and Leus [14] also discuss some recent advances in *sensitivity analysis* for project scheduling. The method developed in this paper can be categorised under the heading *robust scheduling*. Davenport and Beck [10] distinguish between *proactive* and *reactive* approaches. Proactive approaches include the insertion of *redundancy* (typically extra buffer time) into the schedule, the use of *probabilistic techniques* to determine robust schedules and the creation of *multiple schedules* to deal with contingencies.

Although genetic algorithms and meta-heuristics in general are often and successfully used to tackle deterministic scheduling problems, their application to scheduling under uncertainty is very limited. In this paper, we argue that meta-heuristics can very easily be adapted to the require-

ments of a stochastic problem formulation. Using meta-heuristics for this type of optimisation has several advantages that will be discussed later. That there is a need for robust meta-heuristic optimisation is recognised in the influential book "Robust discrete optimisation" [17], when the authors say on p. 354: "*We believe that considerable more effort should be spent in systematic development of [...]* metaheuristic frameworks*, which with minimal adjustment effort can be applied to a large class of robust optimisation problems [...]*". Previous work on robust genetic algorithms can be found in Branke [3, 4], Tsutsui [26], Tsutsui and Ghosh [27], Tsutsui et al. [28], Tsutsui and Jain [29], but the applications of this work are limited to the optimisation of continuous mathematical functions.

A genetic algorithm for robust job shop schedules is developed by Leon et al. [19]. This algorithm uses an objective function that includes some measure of robustness. In their problem, the authors assume that machine operations can be disrupted and that the disrupted operations are restarted immediately after the end of the disruption period (a *right-shift reactive policy*). Robustness is defined as a function of the schedule delay, i.e. the difference between the actual makespan and the projected makespan. A number of different measures, specifically developed to estimate this measure of robustness is developed.

Artificial immune systems have been recently developed [12, 13] as a tool for scheduling under uncertainty. The author's goal is to develop off line a number of partial schedules that can be used as building blocks to quickly compose a good schedule when unexpected events occur. The underlying assumption is that unexpected events (such as machine breakdowns) and the desired reaction of the scheduling system, are to some extend predictable. The partial schedules found by the GA therefore constitute a specific piece of domain knowledge, that can be used when unexpected events occur.

Jensen [16] introduces several novel ideas. In *neighbourhood-based robustness*, the basic idea of which is that if a small set of schedules close to the preschedule is known to be of good quality before an unexpected event occurs, then perhaps one of these schedules provide a good solution in case an unforeseen event occurs. *Co-evolution* is a term to describe genetic algorithms in which a population of robust solutions and a population of machine breakdowns evolve simultaneously. At the end, the first population contains the most robust solutions and the second one the worst machine breakdowns.

3 Single machine scheduling problem formulation

The objective of the scheduling problem in this paper is to schedule a given set of jobs on a single machine without preemption. Every job has a *release*

date that represents the time at which production or assembly of the batch can begin at the earliest. The *processing times* of the jobs are known, as well as their *due dates*. Depending on its importance, a *weight* is attached to every job. A job is called *late* if its completion time is greater than its due date. The objective is to find a schedule of jobs that *minimises the total weighted number of late jobs*. Late jobs are arbitrarily moved to the end of the schedule, as the amount of lateness is considered unimportant. Table 1 clarifies the notation used.

Description	Symbol	Remarks
Release date	r_j	
Processing time	p_j	
Due date	d_j	
Weight	w_j	
Starting time	t_j	$r_j \leq t_j$
Completion time	C_j	$C_j = t_j + p_j$
Lateness status	U_j	$U_j = 1$ iff $C_j > d_j$ and 0 otherwise

Table 1: Single machine scheduling problem notation

This problem is denoted in the standard classification as $1|r_j|\sum w_j U_j$. The problem is $\mathcal{NP}$-hard in the strong sense [18]. For the static case, previous work on this problem can be found in Sevaux and Dauzère-Pérès [24], Dauzère-Pérès [7], Dauzère-Pérès and Sevaux [8], Dauzère-Pérès and Sevaux [9], Baptiste [1] and Baptiste et al. [2].

Although all parameters of the problem can be made stochastic, we only model the case of stochastic release dates. We will show however, that other stochastic parameters can easily be entered into the problem formulation and solved by the modified GA.

4 Robustness

Real-world applications are typically characterised by the absence of certainty about the problem data. This fact should be taken into account when developing the problem formulation. As a direct consequence, the solution algorithm should be able to deal with problems of a stochastic nature in an efficient way. A solution that is relatively insensitive with respect to changes in the problem data it is based on, is called *robust*. In [25], two types of robustness are distinguished: *quality robustness* and *solution robustness*.

A solution is called *quality robust* if the *quality* of this solution is relatively insensitive to changes in the problem data. This type of robustness can also be called *robustness in the objective function space*. The quality robustness of a solution is a measure of how well this particular solution will

perform in a changing environment. While this can be partially captured by estimating how well the solution will perform on average, it is clear that the risk preference of the decision maker should also be taken into account. Some decision makers might prefer a solution that has a high expected quality, but an important risk of not reaching this expected performance. Others might prefer a solution that has a lower expected quality, but is relatively certain not to perform badly. An extremely risk-averse decision maker might even prefer the solution that has the best worst-case performance, independent of its expected quality. Any decision procedure should provide some means of estimating the risk associated with a specific solution.

When problem data changes occur and the current solution does not satisfy the requirements of the decision maker, the problem is re-optimised. If a good solution procedure is used, the quality of the newly found solution is ensured. However, in many cases it is important for the new solution to be "close" to the original (*base-line*) solution. As an example, consider a manufacturer that has a weekly production schedule (i.e. the same set of batches are produced every week). In many cases, a manufacturer that has a fixed weekly schedule will be hesitant to completely overthrow his current schedule because of relatively small changes in the release dates of some production batches. When re-optimising the schedule, this manufacturer will require the new schedule to be as close as possible to the previous one.

Given a base-line solution, a solution found by the solution algorithm is *solution robust* if it is close to the base-line solution. This type of robustness can also be called *robustness in the solution space*.

5 A genetic algorithm for a single machine scheduling problem

A schematic representation of the genetic algorithm used in the rest of the paper, is in algorithm 1.

In this GA, a solution is encoded as a permutation of the n jobs. An initial population is created by randomly generating permutations of jobs. The main loop of the GA is stopped after a fixed number of iterations without improvement of the best solution.

Two parent solutions are drawn from the population according to a *ranking selection* method which give a higher probability to the best individuals [22]. The population is sorted in ascending order of fitness values (ascending order of the objective function), so the best individual is ranked first. The selection of the first parent is made using the probability distribution $2(N+1-k)/(N(N+1))$ where k is the k^{th} chromosome in the ascending order of fitness values, and N the size of the population. Using

Algorithm 1 Basic incremental genetic algorithm

1: *generate* an initial population
2: **while** stopping conditions are not met **do**
3: *select* two individuals
4: *crossover* the two individuals
5: *mutate* offspring under probability
6: *evaluate* offspring
7: *insert* offspring under conditions
8: *remove* an individual under conditions
9: **end while**
10: *report* results

this technique, the median chromosome $(N+1)/2$ has the probability $1/N$ of being selected, while the best individual has the probability $2/(N+1)$ (approximatively twice the median). The second parent is randomly selected with an uniform distribution.

The crossover operator is a one-point crossover operator X1. A crossover point is randomly chosen. The first part of parent P1 is copied into the offspring. The rest of the chromosome is filled by reading the information of the second parent P2. The jobs that are not already in the offspring are copied, preserving the permutation property. See figure 1.

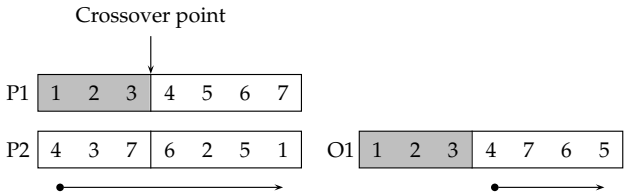

Figure 1: Crossover operator X1

A *general pairwise interchange* mutation operator permutes two jobs randomly chosen in the chromosome. See figure 2. This mutation is done under a probability P_m. For the deterministic case, we chose $P_m = 0.25$.

Figure 2: Mutation operator GPI

After mutation, the solution is evaluated. The fitness value of a solution

is equal to the sum of the weights of the late jobs, i.e.

$$f(x) = \sum_{j=1}^{n} w_j U_j \tag{1a}$$

where

$$U_j = \begin{cases} 1 \text{ iff } C_j > d_j \\ 0 \text{ otherwise.} \end{cases} \tag{1b}$$

The genetic algorithm uses incremental replacement, i.e. the population size remains the same throughout the run. A new offspring is inserted if it improves at least the worst solution in the population. When a new solution is inserted, a solution randomly chosen from the worst half of the population, is discarded.

In Sevaux and Dauzère-Pérès [24] different variations of this simple genetic algorithm, combined with a local search technique are compared to and found to be superior over other methods. The GA used in this paper does not attempt to locally improve solutions using a local search operator. Experimental results show that, when using the GA to find robust solutions, the use of a local search improvement operator tends to overemphasise solution quality. Finding robust solutions requires the exploration of a diverse set of solutions, covering a portion of the search space that is as large as possible.

6 A genetic algorithm for robust schedules

In the following, we describe how the genetic algorithm for the deterministic scheduling problem can be modified so that it finds solutions that are quality robust and/or solution robust. As in [25], this is done by replacing the fitness function of the GA by a so-called *robust evaluation function*. When we want to make the distinction, the fitness function for the deterministic problem is referred to as the *ordinary evaluation function*.

6.1 Quality robustness

Let x be a solution of the problem (a permutation of the jobs). The quality of x is computed by an evaluation function $f(x)$. When we want to indicate that f has parameters, we write $f(x, P)$, where P is the set of problem data. In our case, P represents the characteristics of the jobs (r_j, p_j, d_j, w_j). To allow the GA to find robust solutions, the evaluation function $f(x)$ is replaced by a *robust evaluation function* $f^*(x)$. The robust evaluation function for quality robust solutions adheres to the following principles [25]:

Principle 1 : Each solution is implemented on a modified set of characteristics $S_i(P)$. S is a *sampling function*, that takes a random sample from the stochastic elements of P. $S_i(P)$ is the i-th set of sampled parameters of P. We call the implementation of a solution on a modified set of characteristics a *derived solution*.

Principle 2 : Several evaluations of a solution x on a sample of P are combined into a new evaluation function. An evaluation of a derived solution is called a *derived evaluation*. This new function is the *robust evaluation function* $f^*(x)$.

A possible form of a robust evaluation function is a weighted average of m derived evaluations:

$$f^*(x) = \frac{1}{m} \sum_{i=1}^{m} c_i f(x, S_i(P)) \tag{2}$$

where c_i is a weight associated to this derived evaluation according to its importance and m is the number of derived solutions to evaluate.

A more conservative robust evaluation function examines the worst-case performance of a solution across all derived evaluations:

$$f^*(x) = \max_{i=1...m} f(x, S_i(P)) \tag{3}$$

if f is a minimisation function or

$$f^*(x) = \min_{i=1...m} f(x, S_i(P)) \tag{4}$$

if f is a maximisation function.

For the scheduling problem considered in this paper, we assume the release dates to be stochastic. The robust evaluation function evaluates each solution several times, each time on a new random instantiation of the release dates. The same permutation of the jobs is kept but the release dates are modified. Afterwards, the engine which computes the objective function value of the permutation is called and executed with this new instantiation of the problem. The derived evaluations are combined into a single robustness measure as indicated. This implies that a standard evaluation has to be performed several times for each solution, which increases the computational time. The advantage of this approach however is that the standard evaluation function does not have to be modified. The implementation of a robust evaluation function is therefore very easy and only requires a standard evaluation function to be available and a metaheuristic already running for problems with static data. The only change made is to the evaluation step of algorithm 1, rendering this way of finding robust solutions extremely easy to implement and very flexible.

6.2 Solution robustness

Assuming an initial (baseline) solution x_0 is provided, robustness in the solution space can be obtained by measuring the distance (or similarity) between the solutions found by the GA and the baseline solution. This can be done using a distance measure like the *edit* distance. This measurement is very easily incorporated into the GA. The GA generates many solutions during its search process and it suffices to measure the distance of each encountered solution to the base-line solution, and to record these distances together with the value of the evaluation function. If a solution is required that is both solution-robust and quality robust, the robust evaluation function is used. If only solution robustness is required, the ordinary evaluation function can be recorded.

Of course, optimising only the solution robustness is meaningless since the optimal solution according to this criterion is x_0 itself. A multi-objective decision making process therefore has to be used to choose a solution that simultaneously minimises the distance to the base-line solution and the (ordinary or robust) evaluation function value.

Edit distance First developed in the context of correcting binary codes transmitted across a faulty channel [20], the edit distance is a measure of the similarity between two strings, composed of characters of a finite alphabet Σ. An extra null-character Λ is also defined to signify the absence of a character. Three *elementary edit operations* are defined ($x, y \in \Sigma$): *insertion* ($\Lambda \rightarrow x$), *deletion* ($x \rightarrow \Lambda$) and *substitution* ($x \rightarrow y$). An edit transformation between strings s and t is a set of elementary edit transformations that transform s into t.

Simplifying, the edit distance can be defined as the number of edit operations that is required to transform a *source* string s into a *target* string t[1]. An algorithm to compute the edit distance, due to Wagner and Fischer [31] is given in algorithm 2.

The time complexity of this algorithm is $O(|s| \times |t|)$ where $|s|$ and $|t|$ are the lengths of strings s and t respectively. More efficient algorithms have been developed (e.g. by Ukkonen [30]), but these are beyond the scope of this paper. Under some conditions [31], the edit distance is a *metric*.

Distance between two schedules If all jobs of the scheduling problem are represented by a different character, a schedule can be represented as a string of jobs. The distance between two solutions can be calculated by

[1] In a more general formulation of the edit distance, a distance function $\gamma(x, y) \geq 0$ is defined that determines the non-negative real-valued cost of transforming character x into character y ($x, y \in \Sigma \cup \Lambda$). The cost of an edit transformation is defined as the sum of the costs of the elementary edit operations it contains. The edit distance now is defined as the minimum-cost edit transformation that transforms s into t.

Algorithm 2 Calculation of the edit distance

```
initialise: n ← |s|, m ← |t|
if n = 0 then return m, exit
if m = 0 then return n, exit
Construct a matrix d of size (n + 1) × (m + 1)
initialise ∀i ∈ [0, n] : d(i, 0) ← i, ∀j ∈ [0, m] : d(0, j) ← j
for i = 1 to n do
    for j = 1 to m do
        if s[i] = t[j] then
            c ← 0
        else
            c ← 1
        end if
        d[i, j] ← min(d[i − 1, j] + 1, d[i, j − 1] + 1, d[i − 1, j − 1] + c)
    end for
end for
return d(n, m)
```

the edit distance and interpreted as the number of "changes" that have to made to the first schedule to turn it in to the second one. A "change" can be either an insertion of a job into the schedule, a deletion of a job or the substitution of a job by another one.

Other distance measures for permutation problems Other measures of distance have been proposed for permutation-type problems. Campos et al. [5], for example, propose the following measure for A-type permutation problems (i.e. permutation problems in which the absolute position of the elements is the most important, such as scheduling problems):

$$d(p, q) = \sum_{i=1}^{n} |p_i - q_i|.$$

All solution elements are represented by a rank number, p and q are strings of rank numbers and p_i is the rank number of the i-th element of p. This particular measure has several undesirable properties, the most important one probably being the fact that the distance between solutions is highly dependent on the ranking of the items[2]. For so-called R-type permutation

[2]**Example:** suppose permutations of elements `A`, `B` and `C` are compared. Let $p =$ `ABC`, $q =$ `BAC` and $r =$ `CBA`.
If the items are labelled `A` $\rightarrow$ `1`, `B` $\rightarrow$ `2` and `C` $\rightarrow$ `3` then $d(p, q) = d(\texttt{ABC}, \texttt{BAC}) = d(123, 213) = 1 + 1 = 2$ and $d(p, r) = d(\texttt{ABC}, \texttt{CBA}) = d(123, 321) = 2 + 2 = 4$.
On the other hand, if the strings are labelled `A` $\rightarrow$ `1`, `B` $\rightarrow$ `3` and `C` $\rightarrow$ `2` then $d(p, q) = d(\texttt{ABC}, \texttt{BAC}) = d(132, 312) = 2 + 2 = 4$ and $d(p, r) = d(\texttt{ABC}, \texttt{CBA}) = d(132, 231) = 1 + 1 = 2$.

problems, i.e. problems for which the relative position of the elements is more important, Campos et al. [5] propose the following measure:

$$d(p,q) = \text{number of times } p_{i+1} \text{ does not immediately follow } p_i \text{ in } q.$$

Again, this distance measure shows some serious defects[3]. The edit distance, used in this paper, does not have these drawbacks and is therefore—in our opinion—better suited to calculate the distance between solutions that are represented as a permutation of items.

6.3 Incorporating risk preference

The function $f^*(x)$ estimates the average performance or the worst case performance of a solution, given that some of the parameters of the problem are stochastic. Clearly, the worst case performance measure will lead to solutions that are more conservative. Solutions found using this form of the robust evaluation function will hedge only against the worst possible incidence, independent of the probability that this scenario will occur. This type of robust evaluation function can be used by extremely risk-averse decision makers.

A more subtle manner to incorporate the risk preference of the decision maker, is to include into the robust evaluation function an estimate of the probability that the quality of a solution will deviate from its expected value. A possible measure is the standard deviation of the quality of a given solution over all samples:

$$\sigma^*(x) = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}[f(x,S_i(P)) - f^*(x)]^2}.$$

The two measures can be integrated in a multi-objective decision making approach. A possible way is to find that minimises $f^*(x) + \gamma\sigma^*(x)$, where γ is a parameter indicating the risk-averseness of the decision maker. A more advanced way is to retain all efficient solutions and choose one according to a multi-objective decision making method.

In the first case, $d(p,q) < d(p,r)$; in the second case, this relationship is reversed. This is clearly an undesirable property of this distance measure.

[3]Consider e.g. $p =$ `123456789`, $q =$ `987654321` and $r =$ `214365879`. Clearly, the distance between p and q should be larger than that between p and r as the items in q are much further from their original positions in p than the items in r. Nevertheless, $d(p,q) = d(p,r)$ according to this measure.

7 Application to just-in-time scheduling

7.1 Just-in-time scheduling

Many manufacturing companies have switched (at least partially) to a *just-in-time* policy over the last two decades. In this philosophy, suppliers are required to deliver their raw materials or semi-finished products to the manufacturer's plant right on time for these parts or materials to be taken into production [6]. The suppliers are required to locate a supply hub (e.g. a warehouse) close to the manufacturer's production facility and deliver parts or materials to the manufacturing plant several times a day, at the request of the manufacturer. Based on the *production sequence*, i.e. the order in which the different product batches are be produced in the manufacturing plant, release dates and amounts to deliver of the parts or materials are fixed by the schedulers in the manufacturing plant and communicated to the suppliers. The production sequence determines when the production of each batch begins, and consequently which batches will be finished on time and which not.

Just-in-time delivery promises several advantages, the most important ones being reduced costs and shorter lead times. A negative side-effect from the use of just-in-time however, is the fact that any delay in the delivery by the suppliers will postpone the production of a batch which will lead to longer and unpredictable lead times of the finished product and increase the number of late deliveries. Although manufacturing companies and their suppliers are working hard to reduce the delay of parts or materials, unexpected events can have a significant effect on them. This in turn will negatively influence the number of production batches that are finished on time.

While manufacturers go to great lengths to reduce the number of late deliveries of parts and materials as well as the amount of time with which they are delayed, some factors remain beyond their control. Traffic jams, truck breakdowns, strikes etcetera, are factors that may delay delivery, but are very difficult to control. To deal with these types of unexpected events, a manufacturing company can attempt to design a *robust production sequence*. A quality robust production sequence can be defined as a production sequence of which the quality is relatively insensitive to perturbations in the input data. The quality of a certain schedule can be measured by various quality measures, e.g. by counting the number of batches that are finished on time. A solution robust production sequence may be defined as a production sequence that is "close" to a base-line production sequence that is currently in use by the manufacturer. As mentioned, changing the production sequence drastically when unexpected events occur may increase the quality, but might cause a large disruption in the regular production routine. This in turn may lead to production errors, and other undesirable

effects.

7.2 Modelling a just-in-time production system as a scheduling problem

A just-in-time production environment can be modelled as a scheduling problem, in which each of the production batches is represented by a job that has to be scheduled on a single machine. Solving the problem yields a production sequence that minimises the weighted number of late batches. Of course, other objectives can be used, such as the weighted tardiness.

The following paragraphs describe the problem that a real French assembly factory in the car industry faces. For reasons of confidentiality, real data were not given, but enough information was provided to allow us to create a realistic simulation of reality.

In the assembly factory, we observe a huge number of elementary tasks that we aggregate in a reduced number of aggregated jobs. The number of jobs n to be processed each day varies between 20 and 80. The jobs are of a large variety in nature and have widely spread-out processing times. Processing times for the different jobs are approximately uniformly distributed, with a high variability. Jobs are planned each day, but the schedule approximately repeats itself every week. Past observations show that a significant part of the jobs cannot start at their expected release date because of late deliveries of subcontractors. The percentage of jobs that is delayed is about 20%. Jobs almost never start early. Delays are approximately uniformly distributed between 0 and some parameter 20.

Problem instance generator – deterministic problem From these data, we have created a problem instance generator that generates instances according to the following rules. A single day of 80 five-minute periods is considered. The probability of release dates to arise in the morning is greater than in the afternoon (and the reverse for due dates). Hence release dates are generated according to a gamma law ($\Gamma(0.2, 4)$, which gives a mean of 20 and a standard deviation of 10) and due dates are generated according to the same law but considering the horizon from the end. If $r_i \geq d_i$ the associated job is discarded. To generate only feasible jobs, processing times are uniformly generated in the interval $[1, d_i - r_i]$. Weights are uniformly generated in the interval $[1, 10]$. Twenty different instances are generated for each value of n and n takes its value in the set $\{20, 40, 60, 80, 100\}$. This new set of instances will be called `ODD` for "One Day Data". Each file is numbered `ODD`n_i where n is the number of jobs and i the ith instance for the specified number of jobs (`ODD20_4` is the fourth instance with 20 jobs).

The rules of the problem instance generator are summarised in table 2. $\mathcal{U}(l, u)$ indicates a uniform distribution with lower and upper boundary l and u respectively. $\Gamma(a, b)$ indicates a gamma distribution.

Parameter	Value
Problem size	(20,40,60,80,100)
Total time T	80
Release date r_j	$\Gamma(0.2, 4)$
Due date d_j	$T - \Gamma(0.2, 4)$
Processing time p_j	$\mathcal{U}(1, d_i - r_i)$
Weight w_j	$\mathcal{U}(1, 10)$

Table 2: Deterministic problem instance generator rules

Robust evaluation function In the robust evaluation function, each solution is evaluated a fixed number of times on a modified instance of the problem data. At each evaluation, a number of jobs is randomly chosen and the release dates of these jobs are increased. The m evaluations are averaged to determine the value of the robust evaluation function. The parameters of the robust evaluation function are shown in table 3.

Parameter	Value
Percentage of jobs with delayed release date	20%
Release date delay	$\mathcal{U}(0, 20)$
Number of evaluations per robust evaluation (m)	100

Table 3: Robust evaluation function parameters

The number of evaluations m needs to be sufficiently high to avoid that a non-robust solution is accidentally chosen. Some experiments show that a value of $m = 100$ gives adequate results, while keeping the increase in computation time within acceptable limits. The robust evaluation function value is stored and used for population management purposes (see section 5).

8 Numerical computations

To test the efficiency of the proposed method, a simulation procedure is used. The standard GA, using the ordinary evaluation function, is first run on the original data. The result obtained is called the *standard sequence*. The robust GA is also run with the systematic application (i.e. for evaluation of the solution and population management) of the robust evaluation function. The result obtained is called the *robust sequence*. Once these two sequences are obtained, 1000 replications of the problem instance with randomly modified data according to the disturbances are evaluated through the sequences. Figure 3 summarises the simulation procedure. Results are analysed below. In the sequel, the standard GA procedure will be denoted by the acronym SGA and the robust GA by RGA.

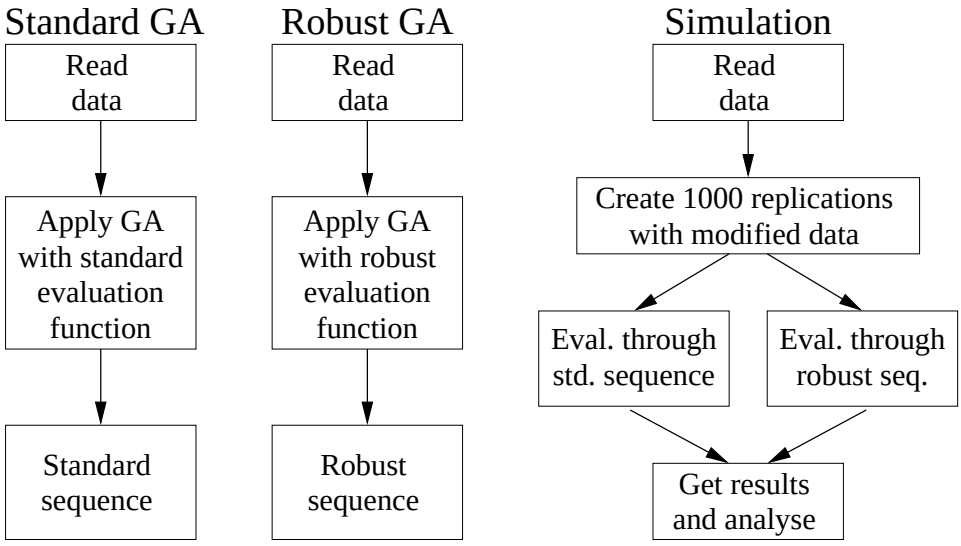

Figure 3: Simulation procedure

Table 4 gives the detailed results for the SGA for the subset of instances with 80 jobs. the column 'CPU' gives the CPU time of the SGA in seconds. The number of iterations is mentioned in the column '#It'. The next two columns respectively show the fitness value of the best individual and the average fitness value of the final population. The last two columns are the results of the simulation with the 1000 replications of the problem with modified instances. The sequence of the best individual is used to evaluate these 1000 replications and the average value is reported in the 'Av. 1000r' column. Column 'Inc.' indicates the increase (in %) of the 1000 modified instances compared to the best solution (e.g. for instance `ODD80_1`, a modification in the release dates could lead to an increasing of 7.54 % of the objective function value, on average).

Similar results are reported in Table 5. In addition we have the robust fitness value (column 'Robust Fit.') which was used during the search to guide the RGA. This value is used as a reference instead of the Fitness value. Note that the average population value was computed on the fitness values and not on the robust fitness values.

For a better comprehension, let us examine in detail instance `ODD80_20`. The best objective value obtained by the standard GA is 365 after 12365 iterations in 0.63 seconds. If we keep this sequence knowing that release dates can be modified, we could expect an objective value of 391.88 (on average) which is an increase of 7.36% of the cost. The Robust GA gives a sequence with a solution of 370 (or more likely with 371.78, the robust fitness value) after 23057 iterations in 151.64 seconds. If now we use this

Name	Cpu (s)	# It.	Fitness	Av. Pop.	Av. 1000r	Inc. (%)
ODD80_1	0.87	16775	400	405.05	430.17	7.54
ODD80_2	1.24	31903	349	353.90	373.00	6.88
ODD80_3	0.64	15606	348	354.19	371.25	6.68
ODD80_4	0.64	14505	411	417.14	431.00	4.87
ODD80_5	1.29	25518	307	312.76	337.28	9.86
ODD80_6	0.52	13217	329	332.62	340.99	3.64
ODD80_7	0.96	20278	331	336.86	361.88	9.33
ODD80_8	0.70	12646	354	357.67	368.84	4.19
ODD80_9	0.99	18863	317	321.43	343.06	8.22
ODD80_10	0.69	14645	344	347.67	366.58	6.56
ODD80_11	0.73	17557	394	398.19	417.80	6.04
ODD80_12	0.75	15224	363	374.52	385.68	6.25
ODD80_13	0.55	10500	317	322.81	338.57	6.81
ODD80_14	0.54	10104	364	368.86	389.80	7.09
ODD80_15	0.65	11040	369	373.86	385.32	4.42
ODD80_16	0.52	10418	370	375.90	384.52	3.92
ODD80_17	0.52	10982	325	331.48	342.11	5.26
ODD80_18	0.84	13599	307	312.81	324.60	5.73
ODD80_19	0.76	15434	357	363.43	376.29	5.40
ODD80_20	0.63	14365	365	372.33	391.88	7.36

Table 4: Results of the Standard GA for 80 jobs

Name	Cpu (s)	# It.	Fitness	Robust Fit.	Av. Pop	Av. 1000r	Inc. (%)
ODD80_1	158.33	22058	406	406.36	411.76	417.60	2.86
ODD80_2	72.52	9956	357	358.91	361.90	366.73	2.72
ODD80_3	108.76	15687	369	363.37	365.62	374.23	1.42
ODD80_4	152.49	20818	429	422.61	424.57	426.65	-0.55
ODD80_5	224.35	29705	310	314.94	317.48	328.88	6.09
ODD80_6	100.98	15109	327	328.24	335.24	335.49	2.60
ODD80_7	147.44	20249	342	345.39	344.33	358.54	4.83
ODD80_8	96.03	11853	358	360.67	363.67	365.09	1.98
ODD80_9	120.26	14271	325	327.12	330.67	337.71	3.91
ODD80_10	126.65	16910	353	355.95	359.57	358.47	1.55
ODD80_11	119.67	15989	403	408.82	414.05	419.97	4.21
ODD80_12	94.76	13801	369	371.24	375.29	377.26	2.24
ODD80_13	62.12	8559	341	333.90	337.33	340.49	-0.15
ODD80_14	85.73	12212	381	378.84	379.76	383.52	0.66
ODD80_15	146.25	16869	373	377.04	382.81	378.48	1.47
ODD80_16	106.09	14472	377	370.44	377.38	380.52	0.93
ODD80_17	144.83	19637	329	329.30	332.00	333.23	1.28
ODD80_18	120.86	15289	313	316.72	315.86	321.17	2.61
ODD80_19	158.79	21159	364	363.40	363.81	368.20	1.16
ODD80_20	151.64	23057	370	371.78	376.43	377.56	2.04

Table 5: Results of the Robust GA for 80 jobs

sequence for the modified data, the expected objective value will be 377.56 (on average) which is an increase of only 2.04%. The solution given by the RGA is then more *quality robust*.

Table 6 gives the summarised results for the whole set of instances from 20 to 100 jobs.

Number	Gap to best solution		CPU time (s)	
of jobs	Std GA (%)	Rob GA (%)	Std GA	Rob GA
20	11.95	3.84	0.03	1.73
40	8.47	2.89	0.13	10.27
60	7.35	3.08	0.36	45.81
80	6.30	2.19	0.75	124.93
100	5.32	2.01	1.53	215.69

Table 6: Deviation from the best solution of SGA and RGA

The results confirm that it is always better to take the stochastic nature of some of the data into account when determining a schedule. On average, modifications of the problem data have a much smaller effect on the objective function value of solutions found by the RGA. However, since the computation of the robust evaluation function requires the ordinary evaluation function to be used 100 times, CPU times are much larger for the RGA. A possible strategy to lower the computational effort is to use the robust evaluation function only when the standard evaluation of a solution is better than a certain threshold. We did not implement such strategies here as the computational times recorded for the RGA are still very reasonable.

9 Helping the decision maker

9.1 Risk analysis

As mentioned in Section 6.3, choosing a robust solution is not an easy task and can be done through a *multiple objective decision making process*. In Figure 4, for a 20 job instance, we plot all the solutions (including the initial solutions) in the space "Standard Deviation of the robust value / Robust Value". The initial solutions are spread out in the space and all the other solutions seem to converge toward a good robust fitness value (to the bottom of the graph). At the same time, the standard deviation of the solutions decreases meaning that the solution are more and more robust. The figure on the right shows only the non-dominated solutions (Pareto solutions) that can be suggested to the decision maker.

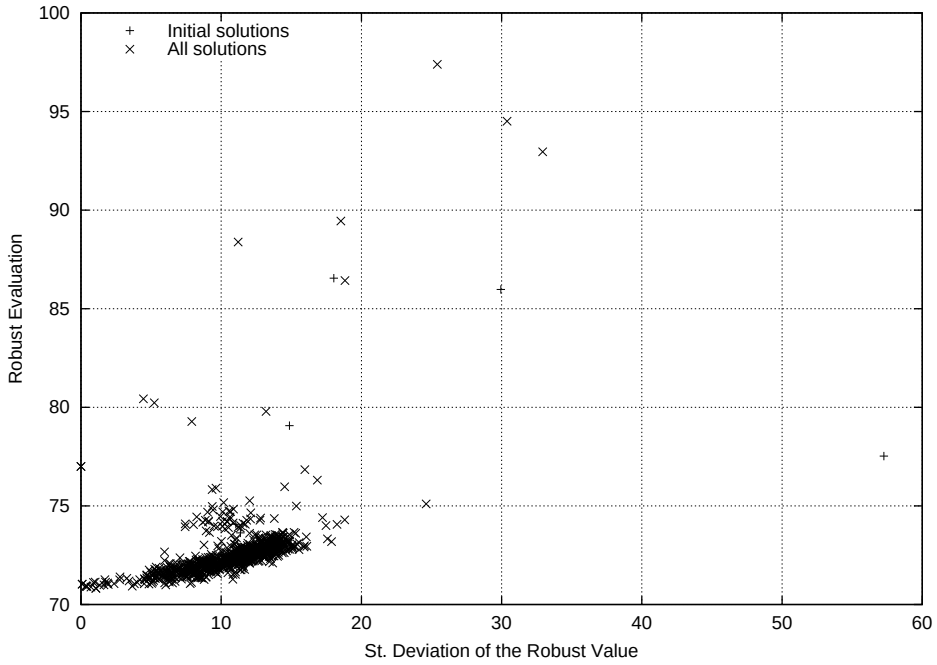

Figure 4: Standard deviation of the robust value—all solutions

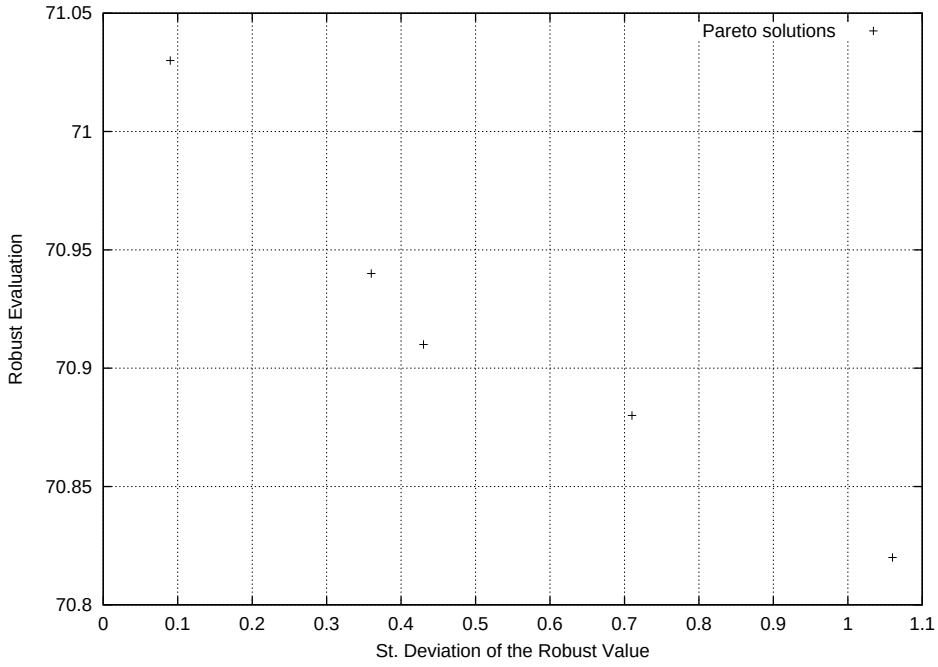

Figure 5: Standard deviation of the robust value—efficient set

9.2 Solution robustness

To obtain solutions that are solution robust, the edit distance between a base-line solution and all the solutions found by the RGA is measured. In an experiment, we have taken as base-line solution x_0 the best solution obtained with the SGA. For the same 20-job instance, figure 6 represents the initial, intermediate and final solutions obtained by the RGA. The number of intermediate solutions can be very large and only non-dominated solutions are kept for clarity.

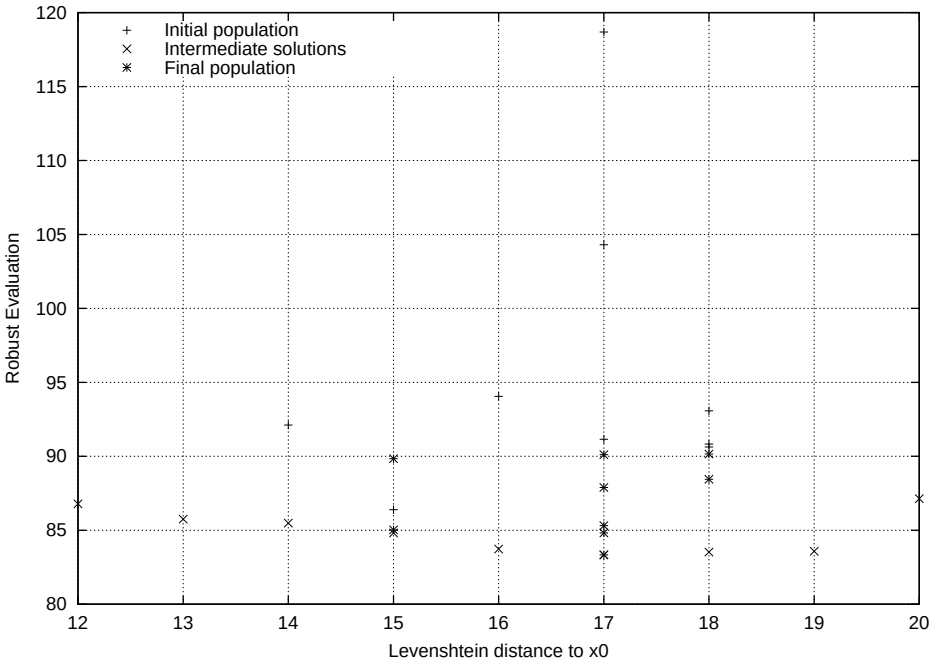

Figure 6: Distance between solutions

For this particular instance, the minimum distance to x_0 that can be obtained by the GA is 12, indicating that all robust solutions are relatively "far" from the base-line solution. Using the information about the distance to the base-line solution and the robustness of a solution, the decision maker can make an informed decision.

10 Conclusion

In this paper, a genetic algorithm for a single machine scheduling problem was modified to find robust solutions. Two types of robustness were defined and it was shown how the GA can be used to find solutions that

are both solution robust and quality robust. It was also shown how information about the risk associated with a specific solution can be incorporated into the decision making process. The modified genetic algorithm was applied to a just-in-time production planning problem. Computational results show that robust solutions can be found by applying some simple modifications of a genetic algorithm for a deterministic problem. This approach is very flexible in that it does not impose any requirements on the number and type of stochastic information that is incorporated. Without changing the solution algorithm in an extensive way, any information that is available about the problem data can be used in the decision making process.

References

[1] Ph. Baptiste. An $\mathcal{O}(n^4)$ algorithm for preemptive scheduling of a single machine to minimize the number of late jobs. *Operations Research Letters*, 24:175–180, 1999.

[2] Ph. Baptiste, L. Péridy, and E. Pinson. A branch and bound to minimize the number of late jobs on a single machine with release time constraints. *European Journal of Operational Research*, 144:1–11, 2003.

[3] J. Branke. Creating robust solutions by means of evolutionary algorithms. In *Parallel Problem Solving from Nature V*, LNCS vol. 1498, pages 119–128. Springer Verlag, 1998.

[4] J. Branke. Reducing the sampling variance when searching for robust solutions. In L. Spector et al., editor, *GECCO 2001 - Proceedings of the Genetic and Evolutionary Computation Conference*, pages 235–242. Morgan Kaufmann Publishers, 2001.

[5] V. Campos, M. Laguna, and R. Martí. Context-independent scatter and tabu search for permutation problems. Working paper, submitted for publication, 2001.

[6] M. Christopher. *Logistics and Supply Chain Management – Strategies for Reducing Cost and Improving Service*. Financial Times Prentice Hall, London, 1998.

[7] S. Dauzère-Pérès. Minimizing late jobs in the general one machine scheduling problem. *European Journal of Operational Research*, 81:134–142, 1995.

[8] S. Dauzère-Pérès and M. Sevaux. Using lagrangean relaxation to minimize the (weighted) number of late jobs. *Naval Research Logistics*, 2002. To appear.

[9] S. Dauzère-Pérès and M. Sevaux. An exact method to minimize the number of late jobs in single machine scheduling. Technical report, Ecole des Mines de Nantes, France, 2001. Submitted.

[10] A.J. Davenport and J.C. Beck. A survey of techniques for scheduling with uncertainty. Unpublished manuscript, 2000.

[11] D.E. Goldberg. *Genetic Algorithms in search, Optimization and Machine Learning*. Addison Wesley, 1989.

[12] E. Hart and P. Ross. The evolution and analysis of a potential antibody library for job-shop scheduling. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 185–202, London, 1999. McGraw Hill.

[13] E. Hart and P. Ross. An immune system approach to scheduling in changing environments. In *GECCO 99, Proceedings of the Genetic and Evolutionary Computation Conference*, pages 1559–1566. Morgan Kaufmann publishers, 1999.

[14] W. Herroelen and R. Leus. Project scheduling under uncertainty – survey and research potentials. Invited paper to be published in the special issue of EJOR containing selected papers from PMS2002, 2002.

[15] J.H. Holland. Adaptation in natural and artificial systems. Technical report, University of Michigan, Ann Arbor, 1975.

[16] M.T. Jensen. *Robust and flexible scheduling with evolutionary computation*. PhD thesis, University of Aarhus, Dept. of Computer Science, Denmark, October 2001.

[17] P. Kouvelis and G. Yu. *Robust Discrete Optimisation and its Applications*, volume 14 of *Nonconvex Optimization and its Applications*. Kluwer Academic Publishers, Dordrecht, 1997.

[18] J.K. Lenstra, A.H.G. Rinnooy Kan, and P. Brucker. Complexity of machine scheduling problems. *Annals of Discrete Mathematics*, 1:343–362, 1977.

[19] V. J. Leon, S. D. Wu, and R. H. Storer. Robustness measures and robust scheduling for job shops. *IIE Transactions*, 26:32–43, September 1994.

[20] V.I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet Physics - Doklady*, 10:707–710, 1966.

[21] M-C. Portmann. Genetic algorithm and scheduling: a state of the art and some propositions. In *Proceedings of the workshop on production planning and control*, pages I–XIV, Mons, Belgium, 1996.

[22] C.R. Reeves, editor. *Modern heuristic techniques for combinatorial problems*. John Wiley & Sons, inc., New York, NY, USA, 1993.

[23] C.R. Reeves. Genetic algorithms for the operations researcher. *Informs Journal on Computing*, 9(3):231–250, 1997.

[24] M. Sevaux and S. Dauzère-Pérès. Genetic algorithms to minimize the weighted number of late jobs on a single machine. *European Journal of Operational Research*, 2002. To appear.

[25] K. Sörensen. Tabu searching for robust solutions. In *Proceedings of the 4^{th} Metaheuristics International Conference*, pages 707–712, Porto, Portugal, 2001.

[26] S. Tsutsui. A comparative study on the effects of adding perturbations to phenotypic parameters in genetic algorithms with a robust solution searching scheme. In *Proceedings of the 1999 IEEE Systems, Man, and Cybernetics Conference (SMC'99 Tokyo)*, pages III–585–591, 1999.

[27] S. Tsutsui and A. Ghosh. Genetic algorithms with a robust solution searching scheme. *IEEE Transactions on Evolutionary Computation*, 1: 201–208, 1997.

[28] S. Tsutsui, A. Ghosh, and Y. Fujimoto. A robust solution searching scheme in genetic search. In H.-M. Voigt, W. Ebeling, I. Rechenberg, and H.-P. Schwefel, editors, *Parallel problem solving from nature - PPSN IV*, volume 10, pages 543–552, Berlin, 1996. Springer.

[29] S. Tsutsui and J.C. Jain. Properties of robust solution searching in multi-dimensional space with genetic algorithms. In *Proceedings of the 2nd International Conference on Knowledge-Based Electronic Systems (KES-98)*, 1998.

[30] E. Ukkonen. Finding approximate patterns in strings. *Journal of Algorithms*, 6:132–137, 1985.

[31] R.A. Wagner and M.J. Fischer. The string-to-string correction problem. *Journal of the ACM*, 21:168–173, 1974.