UNVEILING HIDDEN VALUES OF OPTIMIZATION MODELS WITH METAHEURISTIC APPROACH

Ann Kuo

A DISSERTATION

in

Operations and Information Management

For the Graduate Group in Managerial Science and Applied Economics

Presented to the Faculties of the University of Pennsylvania

in

Partial Fulfillment of the Requirements for the

Degree of Doctor of Philosophy

2014

Supervisor of Dissertation

Steven O. Kimbrough, Professor, OPIM

Graduate Group Chairperson

Eric T. Bradlow, K.P. Chao Professor, Marketing, Statistics and Education

Dissertation Committee

Steven O. Kimbrough, Professor, OPIM

Monique Guignard-Spielberg, Professor, OPIM

Kartik Hosanager, Professor, OPIM

ACKNOWLEDGEMENT

I wish to express my deepest and sincere gratitude to my advisor Prof. Steven O. Kimbrough. Completing this dissertation could not have been possible without his perpetual encouragement, motivation, and support. His insights lead me to this intriguing study and all his contribution of time and ideas make my Ph.D. experience fruitful and stimulating.

I am grateful to my dissertation committee members, Prof. Steven O. Kimbrough, Prof. Monique Guignard-Spielberg, and Prof. Kartik Hosanager for their time, suggestions, valuable comments, and encouragement. I would also like to thank the University of Pennsylvania for the financial support which gave me the precious opportunity and made my Ph.D. work possible with the OPIM program.

It gives me great pleasure in acknowledging the help of Prof. David Harlan Wood; he is always cheerful, enthusiastic, and willing to share his knowledge and experiences.

Finally, I would like to express my heartfelt gratitude, love, and admiration to my parents Prof. T. S. Kuo and A. C. Kuo. I am immensely indebted to them for their upbringing, nurture, unconditional love and endless support in all my pursuits.

ABSTRACT

UNVEILING HIDDEN VALUES OF OPTIMIZATION MODELS WITH METAHEURISTIC APPROACH

Ann Kuo

Steven O. Kimbrough

Considering that the decision making process for constrained optimization problem is based on modeling, there is always room for alternative solutions because there is usually a gap between the model and the real problem it depicts. This study looks into the problem of finding such alternative solutions, the non-optimal solutions of interest for constrained optimization models, the SoI problem. SoI problems subsume finding feasible solutions of interest (FoIs) and infeasible solutions of interest (IoIs). In all cases, the interest addressed is post-solution analysis in one form or another. Post-solution analysis of a constrained optimization model occurs after the model has been solved and a good or optimal solution for it has been found. At this point, sensitivity analysis and other questions of import for decision making come into play and for this purpose the SoIs can be very valuable. An evolutionary computation approach (in particular, a population-based metaheuristic) is proposed for solving the SoI problem and a systematic approach with a feasible-infeasible-two-population genetic algorithm is demonstrated. In this study, the effectiveness of the proposed approach on finding SoIs is demonstrated with generalized assignment problems and generalized quadratic assignment problems. Also, the applications of the proposed approach on the multi-objective optimization and robust-optimization issues are examined and illustrated with two-sided matching problems and flowshop scheduling problems respectively.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF ILLUSTRATIONS

CHAPTER 1 : Supporting Deliberation for Optimization Problems with the Metaheuristic Approach

1.1. Introduction

Many real-life optimization problems cannot be modeled adequately using linear programming. Other techniques, such as integer programming and non-linear programming, are typically used to solve these problems. However, these techniques provide limited views of the solution space and have serious limitations dealing with the stochastic nature of the constraints. In this chapter, we investigate the principles and techniques for providing a systematic, thoroughgoing support for post-evaluation analysis of constrained optimization models that are not (limited to) linear programming models. We introduce the Solutions of Interest (SoI) problem, that of finding non-optimal solutions of interest for constrained optimization models. SoI problems subsume finding feasible solutions of interest (FoI), and infeasible solutions of interest (IoI)[1] . In all cases, the interest addressed is post-solution analysis in one form or another. Post-solution analysis (deliberation) of a constrained optimization model occurs after the model has been solved and a good or optimal solution has been found. At this point, sensitivity analysis and other questions of import for decision-making, discussed in the chapter, come into play and for this purpose the SoIs can be of considerable value. We present examples of two challenging classes of integer programming, the generalized assignment problem (GAP) and the generalized quadratic assignment problem (GQAP). These examples demonstrate how we use SoIs to support deliberation and we report on a systematic approach, using evolutionary computation, for obtaining both FoIs and IoIs.

Structure of the chapter. Section 1.2 presents the constrained optimization problem, post-solution deliberation, our motivation, and the related work. Section 1.3, describes genetic algorithms, the evolutionary-computation-based approach we employ. Section 1.4

[1]Terminology: We shall also use SoI for solution of interest, plural, SoIs; similarly for FoI and IoI.

explores our scheme in the context of benchmark problems for the generalized assignment problem (GAP). Section 1.5 reports the performance comparison of different types of GAs. Section 1.6 compares the effect of two different fitness measures for infeasible solutions. In section 1.7, we propose the SoI search procedure. Section 1.8 demonstrates the usefulness of SoIs, section 1.9 further explores the proposed scheme in the context of benchmark problems for the generalized quadratic assignment problem (GQAP), and section 1.10 contains our concluding remarks.

1.2. Motivation and Related Work

The progress of a field of study can be achieved most fundamentally by advances in solving or gaining better knowledge of the field's outstanding problems. Yet, a field of study may also advance when important new problems are identified and addressed usefully. The contribution of the present chapter is of the second kind. In this chapter we characterize a new, or at least under-investigated, problem pertaining to constrained optimization. We call it the non-optimal solutions of interest (SoI) problem. We divide the SoI problem into two related subproblems, the non-optimal feasible solutions of interest (FoI) problem, and the infeasible solutions of interest (IoI) problem. Further, we demonstrate how evolutionary computation may be used to address these problems, we identify a number of research issues, and we present initial, baseline results on these issues. The subject may be framed as follows.

1.2.1. Constrained Optimization Models (COModels)

The versatility of constrained optimization problems stems from the fact that for most practical problems resources are limited. This scarcity of resources leads to the enormous scope of applications of constrained optimization models in business, management, healthcare, engineering, and basic science. Two major points are worth mentioning here. First, models are mathematical abstractions of the problems. Exactly how accurately the model depicts the actual problem is one question. Second, optimization problems in reality often involve soft constraints. Soft constraints, by virtue of being potentially changeable, imply potential options—this further complicates modeling.

Having been distinguished from their actual optimization problems, the optimization models present two very different challenges for supporting decision-makers. The first challenge is the *solution problem*, which is about finding good, even optimal, solutions to a given model. The second challenge is the post-solution deliberation problem (or in short, the *deliberation problem*), which provides the decision-maker with useful information for making an informed decision for the underlying problem based on the model at hand.

Solution Problem: the optimization problem is the problem of finding an optimal solution to the model. An exactly optimal solution satisfies the constraints of the model and is one where there is no other solution that has a superior objective function value and that also satisfies the constraints. If a solution x is exactly optimal, we denote the exact optimal solution by x^* and its objective value by z^*. Oftentimes exact solution methods are not available or not effective in finding exactly optimal solutions. Heuristics are then used to solve the problem. If a solution x is the best known solution then we say it is heuristically optimal; we denote it by x^+ and its objective function value by z^+.

Much attention has been given to the optimization problem for COModels, in both the operations research (OR) and the heuristics communities. Consequently, the optimization problem becomes central to the subfield of constrained optimization with evolutionary computation (or more broadly, with metaheuristics). Constrained optimization with metaheuristics is a vibrant area; a great deal of progress on solving the optimization problem has been made and is continuing to be made, as evidenced by the hundreds of papers published each year investigating and describing use of evolutionary computation (and metaheuristics generally) to solve COModels.

Deliberation Problem: compared to the optimization problem, however, the deliberation problem has received little attention. Deliberation (post-solution analysis) takes place after a constrained optimization model has been formulated, a solution or evaluation procedure applied, and results therefrom obtained.

While we may specify the cost or profit function for the problem at hand, real-world data used to evaluate it are rarely precise. Furthermore, the weights and different forms of contributions involved are usually arbitrary. These complications diminish the meaning that we try to attach to the "best solution" for our problem. Often, what we truly desire is a "good solution", which we can conclude on a time scale short enough so that the solution can be used in the choice of appropriate action. Most of the time, a "good solution" is more meaningful than a nominally-better "best solution". To reach the conclusion of such "good solution", post-evaluation analysis is required. Before we further dive into the deliberation problem, we first introduce the key element of the deliberation process: the non-optimal solutions of interest, the SoIs.

1.2.2. The Solution of Interest (SoI) Problem

Pertaining to constrained optimization, we call the new (or at least under-investigated) problem the *non-optimal solutions of interest (SoI)* problem. The SoI problem is about finding non-optimal solutions of interest (SoIs) for constrained optimization models. Such non-optimal solutions of interest could be either feasible or infeasible. We call the feasible solutions of interest the FoIs and the infeasible solutions of interest the IoIs.

Roughly speaking, a COModel partitions its solutions into two classes: the feasibles, which satisfy all constraints in the model, and the infeasibles, which each violate at least one constraint. Among the feasibles, what interests us—the FoIs (the non-optimal feasible solutions of interest)—are those feasible solutions that are superior in their objective function values relative to z^{+} and that consume fewer resources than x^{+}. We call the problem of finding the feasibles of interest for a given COModel the *FoI problem.* Among the infeasibles, what interests us—the IoIs (the infeasible solutions of interest)—are those infeasible solutions that are superior in their objective function values relative to z^{+} and that are close to being feasible. We call the problem of finding the infeasibles of interest for a given COModel the *IoI problem*. To sum it up, both the FoI problem and the IoI problem can be classed under the more general SoI problem. We will be more precise in the sequel about the FoIs and the IoIs. First, however, let us see why the SoI problem is interesting and potentially important.

1.2.3. Deliberation Problem

At this stage of the modeling life-cycle a number of questions arise naturally, and for applications, most crucially. The deliberation problem (post-solution analysis), according to Greenberg [1993a], "is [the] probing into the meaning of an optimal solution. This includes conventional questions of sensitivity, and it includes some additional analyses that are unconventional in the sense that they go beyond textbook definitions." Kimbrough and Wood describe the deliberation problem as the assessment processes conducted after obtaining preliminary optimal or heuristically good solutions. Other terminologies used in the literature in place of deliberation problem include: post-evaluation analysis, post-solution analysis, post-optimality analysis and candle-lighting analysis. Nevertheless, the primary goal of deliberation is "to reconsider assumptions made in the model in the light of information generated while finding the good solutions as well as information not previously detailed in the model (Kimbrough and D.H.Wood [2006])."

The deliberation problem considers, at least in principle, actions that might be taken to revise the model's assumptions. These considerations are based on weighing solution results along with knowledge not directly reflected in the model. As stated in Kimbrough et al. [2009a], the deliberation problem arises once a good solution is to hand, call it x^+ with value z^+, for a COModel [constrained optimization model]: Should the best available solution be implemented exactly or should we re-consider the model? Are there profitable opportunities to acquire additional resources and thereby relax one or more constraints? On the other hand are there solutions available inferior to x^+ in terms of z^+, but which would consume substantially less in terms of valuable resources? And so on for other deliberations.

The post-solution deliberation of COModels can being organized around three types of questions: the **what-if** question, the **why** (and **why-not**) question, and the **what-does-it-take** question. Descriptions and examples of each type of question are given as follows.

- With "what-if?" questions one asks about the consequences of changing one or multiple parameter values. Sensitivity analysis falls into this category. Examples: What if constraint 7 is tightened by 5%? What will be the new optimal solution and objective value?

- With "why?" and "why-not?" questions one searches for reasoning to explain the proposed solution. For example: Why was job *a*, instead of job *b*, assigned to a certain processor in the optimal solution? At least part of the answer lies in finding solutions in which job *b* is so assigned and then examining the consequences of such assignment—e.g., a particular constraint being violated because of *b* consuming much more of that resource. (See Greenberg [1993b] for a nuanced discussion of why-questions in a classic OR setting.)

- Finally, with "what-does-it-take?" questions, one looks for potential solutions that can achieve a newly set a goal, such as a higher value of z or freeing up a certain amount of constrained resources. For example: given a maximization problem at optimality z=635. What does it take—what does one need to do—in order to increase the z value by at least 5 units?

From a different point of view, Branley et al. [1997] categorize the common deliberation questions into two main types:
Sensitivity analysis questions:

- Do small changes in the model's parameters have large effects on the objective function value?

- What is the range of changes that will affect the accepted solution?

Option discovery questions:

- Are there advantageous opportunities to change the assumptions of the model? For example:
  - Would a change in the RHS of a constraint yield a significantly improved objective value? If so, does the cost of changing the RHS lead to extra profit?
  - Are there good solutions for which there are large quantity of certain resource leftover? If so, can the slack RHS resource be profitably used for some other purpose?

These are all questions of great practical import in the use of COModels and none of them can be addressed with only the optimal solution on hand. These considerations put two fundamental questions into play. The first is, "Why are the FoIs and IoIs (as characterized above) interesting and useful to have?" Call this the motivation question. The second is, "Given that we are interested in FoIs and IoIs, what are effective and comprehensive ways of finding them?" Call this the technical question. We have already given a short answer to the motivation question: FoIs and IoIs can be used to support, and are what we need to support, post-solution analysis and deliberation with COModels (perhaps excluding linear programming models). We can use the FoIs and IoIs to answer valuable what-if?, why?, and what-does-it-take? questions.

Post-solution analysis has long been recognized in the operations research (OR) and management science community as an important and valuable aspect of applied modeling. (See Greenberg [1993a,b,c, 1994] for a comprehensive discussion from the classical exact solution, OR perspective.) Classical OR (exactly optimal solutions) methods for post-solution analysis are most developed for linear programming models. Although there is important work for integer programming models (see Geoffrion and Nauss [1977] and Greenberg [1997] for reviews), the results tend to be very model-type-specific and of limited scope. Moreover, these methods do not generally apply when the primary solution method is a metaheuristic, as it often is and must be in practice. Therefore, we wish to discuss the deliberation problem, particularly about how metaheuristics may be used effectively to support the deliberation (post-solution analysis), which complements, and in no way conflicts with, the optimization problem or its methods of solution. What follows in the next section begins to address both of the motivation and the technical questions.

1.2.4. Complexity of the Problems

The COModels that we are concerned with, e.g., integer programming models (linear or not), mix-integer programming models (linear or not), are all NP-hard as optimization problems, and in practice will often (but not always, this is not necessary, as we shall see shortly) be approached with heuristic solvers. One approach to obtaining FoIs and IoIs is to alter the COModel's parameters systematically in the neighborhood of the boundary and re-solve the model. To see the problem with this approach, consider only the right-hand-side values of the constraints in a small model, one having just five constraints. Assume we are interested in infeasibles that are within 5 units of violation on each constraint and feasibles that have slack of at most five on each constraint. This implies $(5 + 5 + 1)^5 = 161,051$ different models that would have to be solved. If we are interested in ± 10 units on each side of constraints, we get $21^5 = 4,084,101$ different models to solve. If we start considering other parameters in the model, the problem becomes more complex. Even if the COModel can be solved very quickly by an exact optimization solver, these kinds of numbers are overwhelming.

Clearly, except for very small problems, it will not be computationally feasible to sweep out the FoIs and IoIs by resolving the problem systematically. In general, there are no known methods to generate FoIs and IoIs from a solution (or solutions) obtained by other means.[2] Without a fast (polynomial) method of generating the FoIs and IoIs, our natural approach

[2]There are special cases (linear programming and changes of basis, and some work in mixed integer programming Greenberg [1997]) but they are model-specific and are not responsive to a pre-defined feasible or infeasible region of interest (e.g., infeasible solutions within 5 units of the constraint boundary). Of course, no potentially useful method should be dismissed; these and similar methods merit investigation in the present context of deliberation support and the problem of populating the FoIs and IoIs.

is to sample them efficiently, which leads us to metaheuristics, particularly population-based varieties, since in the process of solving a COModel, population-based metaheuristics sample the solution space in an intelligent and biased fashion. Their biases seek regions of better performance. These regions are typically on or near the boundary of the feasible region (otherwise the constraints are moot). These solutions near the feasible-infeasible boundary also typically turn out to be the FoIs and IoIs.

In the rest of this chapter, we demonstrate how one can utilize the population-based metaheuristic, genetic algorithm (GA), to effectively collect SoIs and support deliberation for standard benchmark instances of the generalized assignment problem (GAP) and the generalized quadratic assignment problem (GQAP). We have accumulated a substantial body of data, much too large for full discussion in a thesis chapter; what we describe here is representative of our body of results. Before presenting our results, we briefly describe the GAP at the end of this section. Because the genetic algorithm (GA)—the metaheuristic we have employed to generate our results—is a common approach for the topics of next two chapters, we will describe GA in greater detail in an individual section, Section 1.3.

1.2.5. Generalized Assignment Problem (GAP)

The GAP is a widely-studied combinatorial optimization problem. Given m agents (or processors) and n tasks (or jobs), the goal is to find the maximum-profit assignment of each task to exactly one agent, subject to the capacity of each agent. The GAP can be formulated as follows. Let $I = \{1, ..., m\}$ index the set of agents and $J = \{1, ..., n\}$ index the set of jobs. The decision variables x_{ij} are set to 1 if job j is assigned to agent i, 0 otherwise.

$$\max_{x_{ij}} z(P,A,b) = \sum_{i \in I} \sum_{j \in J} p_{ij} x_{ij} \tag{1.1}$$

$$\textit{subject to}: \quad \sum_{i \in I} x_{ij} = 1 \qquad \forall\, j \in J \tag{1.2}$$

$$\sum_{j \in J} a_{ij} x_{ij} \leq b_i \qquad \forall\, i \in I, x_{ij} \in \{0,1\} \tag{1.3}$$

The constraints, including the integrality condition on the variables, state that each job is assigned to exactly one agent, and that the capacities of the agents are not exceeded (Kellerer et al. [2004], Martello and Toth [1990]). Matrices **P** and **A** and vector **b** with elements p_{ij}, a_{ij}, and b_i are the parameters of the model. Each inequality in expression (3) is said to represent a constraint (on the corresponding agent) and the b_is are the *right-hand side* (RHS) values.

The GAP is known to be an NP-hard problem that is important in practice and that is prototypical of difficult optimization problems. Exact approaches to solve GAP include branch-and-price (Savelsbergh [1997]), and branch-and-bound (Nauss [2003]) while heuristic approaches include tabu search (Diaz and Fernandez [2001]), and path relinking with ejection chains (Yagiura et al. [2006]).

In solving a GAP we find an (either exactly or heuristically) optimal decision variable setting, x^+, with corresponding objective value $z^+ = z(P,A,b)$. We then consider the solutions and objective values of the problem under modification of the parameters **P**, **A**, and b. As we have seen, it is not practical to alter the parameters and resolve the model, given the scale necessary to do so. Our thought is to use population-based metaheuristics, and evolutionary computation particularly, to populate the SoIs as a by-product of solving the model. We will illustrate further details in section 1.4 where the experiments are recorded.

1.2.6. Generalized Quadratic Assignment Problem (GQAP)

The GQAP is not only a generalization of the quadratic assignment problem (QAP) but also a generalization of the GAP. Categorized as the facilities location problem, the generalized quadratic assignment problem is considered to be one of the most difficult combinatorial optimization problems. It models the following real-life case:

> There are a set of m weighted facilities and a set of n capacitated locations (sites). For each pair of facilities, the traffic/flow (e.g., the amount of supplies transported between the two facilities) is specified. For each pair of locations, the distance between the two locations is specified. For each pair of facility and location, the cost of assigning a facility to a location is specified. The unit traffic cost is given.
>
> The goal is to find the minimum-cost assignment of each facility to exactly one location, subject to the location capacities.

To define the problem, we use the following notation:

- M: a set of facilities. Let $I,\ J = \{1, ..., m\}$ index the set of facilities.
- N: a set of locations (sites). Let $H,\ K = \{1, ..., n\}$ index the set of sites.
- x_{ik}: the decision variables x_{ik} are set to 1 if facility i is assigned to site k, 0 otherwise.
- w_i: the weight (or space requirement) of facility i.
- q_k: the capacity of site k.
- e_{ik}: the cost of installing (assigning) facility i at (to) site k.
- t_{ij}: the traffic intensity from facility i to j.
- d_{kh}: the distance between location k and h, $d_{kh} = d_{hk}$.
- u: the unit travel cost per unit distance and per unit flow volume.

The GQAP is formulated as follows:

$$\min_{x_{ik}} \quad \sum_{i=1}^{m}\sum_{k=1}^{n} e_{ik}x_{ik} + u\sum_{i=1}^{m}\sum_{j=1}^{m}\sum_{k=1}^{n}\sum_{h=1}^{n} t_{ij}d_{kh}x_{ik}x_{jh} \tag{1.4}$$

$$\textit{subject to}: \quad \sum_{k\in N} x_{ik} = 1 \qquad \forall\, i \in M, \tag{1.5}$$

$$\sum_{i\in M} w_i x_{ik} \leq q_k \qquad \forall\, k \in N, \tag{1.6}$$

$$x_{ik} \in \{0,1\} \qquad \forall i \in N,\ \forall k \in M. \tag{1.7}$$

The constraints, including the integrality condition on the variables, state that each facility is assigned to exactly one location, and that the capacity of each location is not exceeded. Matrices $\mathbf{E}$, $\mathbf{T}$, $\mathbf{D}$, and vector $\mathbf{W}$ and $\mathbf{Q}$ with elements e_{ik}, t_{ij}, d_{ij}, w_i, and q_k are the parameters of the model. Matrix $\mathbf{X}$ with element x_{ik} is the decision variable.

1.3. An Evolutionary Computation Based Approach - Genetic Algorithms (GAs)

1.3.1. Genetic Algorithm Overview

First introduced by John Holland in 1975, genetic algorithm (GA) is the origin of the now flourishing field called Evolutionary Computing. Known as Neo-Darwinism, the property of adaptation is often described by the equation:

$$Adaptation = Variation + Heredity + Selection.$$

Variation, which refers to how individuals can differ from each other in a population, is crucial since evolution operates on the entire species, not on an individual. By definition, variation can be expressed only in terms of multiple individuals; thus, parallelism and spatial multiplicity are essential ingredients in the algorithm of evolution. Meanwhile, heredity is a form of temporal persistence. The children inherit traits from their parents in discrete chunks of information (e.g., the famous experiments of Mendel's pea plants); the traits can be iteratively passed down a time line from ancestors to descendants. Thus, we have both parallelism and iteration as fundamental pieces of the biological equation for adaptation. Mimicking biological adaptation, genetic algorithms work with populations of solutions (the parallelism) and attempt to guide the search toward improvement, using a "survival of the fittest" principle (more properly, one should say "survival of the reproducers", since "survival of the fittest" actually equals to "survival of the survivors," in our implementations, Flake [1998]). The quality of each solution is measured by a fitness function, which is usually equivalent to the objective function of the optimization problem. The search proceeds through a number of generations (the iteration) with each individual contributing to the next generation in proportion to its fitness. To imitate the nature selection "survival of the fittest", one can select the mating candidates randomly, using a weighted probability function to reflect actual fitness values, scaled values, or simply the rank of fitness values. The selected mating candidates then go through the reproduction stage and generate the offspring. The evolution continues as the offspring replaces the parent generation and produces the new offspring. A simplified genetic algorithm is outlined in Figure 1.

Other than Holland's pivotal publication (Holland [1998]), Reeves also gives a comprehensive introduction of GA (Glover and Kochenberger [2003]). We briefly list the basic operations for a GA as follows:

- Solution Presentation: Among many possible presentations, in a GA, a potential solution to a problem can be represented as a string of numbers that serve as arguments to an evaluation function.

- Population Initialization: Typically, the initial population in a GA consists of randomly generated solutions (not necessarily feasible ones). It is also common to initialize the population with given solutions.

- Initialize the population of solutions, P, with size N.
- Repeat for a certain number of iterations (the given number of generations):
  - Create an empty population P'.
  - Evaluate the fitness of every individual in the current population P.
  - Repeat until P' reaches size N:
    1. Select two individuals from P based on some fitness criterion.
    2. The selected two individuals mate and produce offspring.
    3. The offspring mutate with a certain probability.
    4. Add the two newly generated individuals (the offspring) to P'.
  - Let P now be equal to P'

Figure 1: Genetic Algorithm Outline

- Fitness Evaluation: For every generation, fitness evaluation is performed on each solution. This is commonly the most expensive operation and is recognized as the standard point of comparison for different methods.

- Candidate Selection: The selection scheme primarily determines the convergence characteristic of GAs (Estivill-Castro [1996]) and is often cited as the main reason in cases where premature convergence halts the success of a genetic algorithm. Various methods have been proposed to select the mating candidates (e.g., Rank-based selection, local selection, truncation selection, tournament selection, Roulette wheel selection, and etc.), but the selection is always done on the basis of the individual's fitness value.

- Generation Perturbation: There are many different ways for perturbation to the next generation members. The simplest means are mutation and crossover.

  - Mutation: Mutation operation is commonly considered as a background operator for GAs. It is mainly used to recover desirable genes that have been accidentally deleted from the population. It is part of the lore for GAs that low mutation rates lead to efficient search of solution space, while high rates result in diffusion of search effort and premature extinction of favorable schemata (gene pattern) in the population. Tate and Smith [1993] point out that this conservative strategy is based primarily on empirical findings using simple string encodings over low cardinality alphabets. The particular features of simple string encoding, which make high mutation undesirable, are absent in many more complex GA implementations.

  - Crossover: Built on the Building Block Hypothesis (BBH), which states that GAs work by discovering, emphasizing and recombining low-order schemata in high-quality strings in a parallel manner, the crossover operator is extensively explored. The commonly used crossover operators include single-point crossover, multi-point crossover (Syswerda [1989], Jong and Spears [1992]), and uniform crossover (Spears [1992]).

- Stopping Condition: The termination condition of the evolution process. Common terminating conditions include:

  - Fixed number of generations or a pre-set computation time is reached.
  - A pre-set fitness value is reached by the fittest individual.
  - The fitness has reached a plateau such that successive iterations no longer produce better results.

GAs have been applied successfully to a wide range of problems. Examples of interesting applications include: R.J. Bauer [1994] and J.Shoaf and J.A.Foster [1998] discover attractive investment strategies with GAs, D.O. Boyer and Perez [2001] use GA as an alternative to least squares estimation for non-linear model parameter estimation, and Pfeiffer [2007] proposes to solve combinatorial auction problems with GAs. For earlier publications, Alander [1995], an indexed bibliography compiled and maintained by Alander, reports over 3000 GA publications between 1957 and 1995 (books, proceedings, journal articles, and Ph.D.

dissertations).

1.3.2. Genetic Algorithms for Constrained optimization Problems

Even though GAs have been used successfully as metaheuristics for optimizations, feasibility constraints on optimization problems pose special difficulties for GAs, primarily because mutation and crossover operators are blind to feasibility considerations. The common treatments for constraint problem can be categorized into four major groups:

- Repairing strategy: Feasibility is restored for infeasible solutions via some repair operator. It has been proven effective for certain constrained optimization problems. However, sometimes the strategy increases the complexity of the problem.

- Operator modifying strategy: To maintain the feasibility of solutions, the genetic operators are modified based on the requirement of the problem at hand.

- Rejection strategy: Only feasible solutions are considered during the search process. If, at any stage, there is no feasible solution, this strategy fails to work.

- Penalty function strategy: The infeasible solutions are penalized proportionally to the size of the constraint violations with a penalty parameter. The tuning of the magnitude of penalty term is critical.

Among these strategies, penalty function strategies are the most commonly used and are proven to be very effective. However, it is difficult to balance the objective function against the degree of constraint violation such that neither is dominant. Although many researchers use adaptive variation of penalty parameters and penalty functions, the general conclusion is that these variations are specific to a problem and cannot be generalized. Consequently, other alternative penalty-free approaches have been suggested. Deb and Agarwal [1999] propose a penalty-parameter-free penalty GA that uses the penalty function to pressure the selection toward feasible region while maintaining diversity among the feasible solutions by avoiding comparison between two solutions that are far away from each other. Coello and Montes [2002] offer a dominance-based selection scheme to incorporate constraints into the fitness function and avoid using penalty function. A good review of the penalty-based methods in GAs for handling constraints is given by Yeniay [2005]. Considerable attention has been paid to treating constraints in constrained optimization problems but no consensus method has emerged. For excellent reviews on constraint handling see Coello [2002], Michalewicz and Fogel [2000], Michalewicz [1995, 1996]. A dedicated Web site by Coello is available (Coello [2008], last updated Dec. 2008).

1.3.3. Feasible-Infeasible Two Population GA (FI-2Pop GA)

In addition to the lack of a forthcoming major rule on defining the proper penalty, there is another more fundamental problem on combining the evaluations for objective function and constraint violations. In constrained optimization, a GA will drive the population of solutions to the neighborhood of the effective Pareto frontier. With a penalty function, infeasible solutions are heavily penalized, leading to likely loss of useful genetic material that places the solutions near the frontier (Kimbrough et al. [2002b]).

Intuitively, if one can separate the measuring of performance and feasibility (e.g., by evaluating feasible solutions with only objective function while measuring the infeasible solutions by its distance to the feasible domain), there may be better chance to find optimal solutions that are located on the boundary between feasible and infeasible spaces.

The FI-2Pop GA is the principled response with such approach (Kimbrough et al. [2002c]). Kimbrough et al. [2008] investigate the behavior of the FI-2Pop GA both theoretically and empirically. The authors also prove that The No Free Lunch (Wolpert and Marcready [1997]) results do not apply to constrained problem classes involving "fixed constraints". Considering a setting with fixed constraints is not unreasonable since many real world problems have constraints that reflect domain structure. The FI-2Pop GA there by escapes the NFL implications. Whether FI-2Pop GA performs better or worse than random search for a fixed constraint problem set is unknown, on average. So far it has been shown as a credible option for solving constrained optimization problems involving fixed constraints such as knapsack problems (Branley et al. [1997], Kimbrough et al. [2002b, 2004, 2002a]) and other problems described by Michalewicz [1996] and GAMS World (GAMSWorld [2013]).

With the FI-2Pop GA, the population is divided into two groups: the feasible solutions, which do not violate any constraint, and the infeasible solutions, which violate at least one of the constraints. The ordinary objective values are calculated for the feasible solutions and assigned as their fitness scores; meanwhile, the infeasible solutions are measured with regard to their distance to the boundary of feasible region. In the selection stage, solutions are compared only with other solutions in its own group. A selected individual then mates with another candidate of the same group to generate offspring. The generated offspring solutions will be kept in two separate pools according to their feasibility. The old groups (both feasible and infeasible) will be replaced with their corresponding new offspring collections. The evaluation–selection–reproduction process repeats with the new generation. An outline of our version of the FI-2Pop GA is available in the experiment section Figure 3.

1.3.4. GA for GAP

Much work in the field of GAs has also been devoted to the GAP. Some proposed GAs are described in the following paragraph. Working with the Minimum GAPs, Chu and Beasley [1997] offer a GA with problem-specific heuristic operator which involves two local improvement steps. After the regular crossover for reproducing offspring individuals, the GA attempts to improve the feasibility and the cost with the heuristic operator. The problem specific GA performs very well with the Beasley OR-Library test sets (Beasley [2009]). Wilson [1993] develops a GA that tries to genetically restore feasibility to a set of near-optimal infeasible solutions. The restoring process continues until one feasible solution is obtained and then improved upon by a process of local searches. The proposed GA generates near-optimal solution rapidly, although not as fast as some older heuristic approaches. Feltl and Raidl [2004] propose a hybrid GA with a modified selection and replacement scheme for handling infeasible solutions and a heuristic mutation operator. Compared to the commercial general purpose branch-and-cut system CPLEX, the results indicate that CPLEX is able to handle relatively large size problems (e.g., problems with 80 constraints and 100 decision variables), although the proposed GA outperforms CPLEX on the largest test instance (80 constraints and 400 decision variables). Another hybrid GA, the Guided GA (GGA), is proposed by Lau and Tsang [1998]. A combination of GA and Guided Local Search, the GGA uses extra weighting operation to identify which genes in a chromosome are more susceptible to being changed during cross-over and mutation. GGA also performs very well with the Beasley OR-Library test sets.

1.4. Experiment Settings for Case Study on General Assignment Problems

1.4.1. Test Problem Settings

To evaluate the performance of two GAs on providing deliberation information, we solve a set of 21 benchmark GAPs which range from instances with 8 agents and 24 jobs to 10 agents and 60 jobs. The 21 problems belong to 5 different problem sets. All problems are publicly available via the internet at the Beasley OR-Library (Beasley [2009]). The basic characteristics of the five problem sets are described in Table 1.

Problem Set Name	# Agents	# Tasks	Optimal Value Range
GAP5	8	24	558-568
GAP9	10	30	706-723
GAP10	10	40	947-963
GAP11	10	50	1139-1195
GAP12	10	60	1433-1451

Table 1: Information on Benchmark Generalized Assignment Problem Sets

1.4.2. The Solution Representation

In this case study, all solutions are presented as a string of integers. The i^{th} number in the solution string indicates the ID of an agent who is assigned to process the i^{th} task. Such solution representation automatically fulfills the requirement that every job must be assigned to exactly one agent. To illustrate this representation, consider the GAP5_1 scenario. GAP5_1 has 8 agents and 24 tasks. The agents have identifiers numbered from 1 to 8. There are 24 numbers on the solution string. A solution string may look like following:

$$7\,4\,3\,2\,6\,1 \quad 3\,6\,8\,4\,2 \quad 1\,6\,4\,8\,7\,7 \quad 1\,4\,2\,3\,6$$

This solution should be interpreted as follows:

- task 2, task 10, task 15 and task 20 are assigned to agent 4
- task 3, task 7 and task 23 are assigned to agent 3
- the only task assigned to agent 5 is task 22
- $\cdots$

1.4.3. The GA Implementations

Two different GAs, the Penalty GA and the Feasible-Infeasible Two Population (FI-2Pop) GA are implemented to solve the benchmark GAPs. For the Penalty GA, during the fitness evaluation, a penalty is calculated in proportion to its Euclidian distance to the boundary of feasible region. The fitness score is assigned as a net score after subtracting the penalty from its original objective value. The selection process is executed based on comparing fitness score over pairs of solutions randomly drawn from the whole population. The winners of the selection stage will join the reproduction phase. The newly generated offspring are then collected as the next generation. The evaluation–selection–reproduction process repeats with new generation. Figure 2 presents the outline of the Penalty GA.

With FI-2Pop GA, the population is divided into two groups - the feasible solutions, and the infeasible solutions. The objective values are calculated for the feasible solutions and assigned as their fitness scores; meanwhile the negative value of the Euclidian distance to the boundary of feasible region is calculated as the fitness scores of the infeasible solutions. In the selection stage, solutions are compared only with other solutions in its group. To

```
• Randomly initialize a population of solutions, P, with population size N.
• GenerationCount = 0.
• while (GenerationCount < maxGenerations):
    – GenerationCount = GenerationCount + 1;
    – P' ⟵ [ ]
    – Evaluate the fitness of every individual in the current population P:
      Evaluate the individual's feasibility first.
        ∗ If individual is feasible: Fitness score = objective value.
        ∗ If individual is infeasible: Penalty is calculated according to the penalty
          function and the distance between the evaluated individual and the bound-
          ary of feasible region. i.e. Fitness score = objective value - penalty.
    – Repeat until P' reaches size N:
        1. Select two individuals from P based on some fitness criterion.
        2. The selected two individuals mate and produce offspring.
        3. The offspring mutate with a certain probability.
        4. Add the two newly generated individuals (the offspring) to P'.
    – P = P'.
```

Figure 2: Pseudocode for the Penalty GA

give equal weight on both populations, half of the child population will be produced by the feasible solution pairs while the other half will be produced by the infeasible solution pairs. A selected candidate mates with only candidate from its group (e.g., feasible candidate mates with feasible candidate). The offspring are collected as the new generation. The evaluation-selection–reproduction process then repeats with the new generation. Figure 3 presents the outline of our FI-2Pop GA.

The GA Operators

The operators implemented in this study include single-point crossover, uniform random mutation and tournament-2 selection.

- Single-Point Crossover: A crossover point is randomly selected. The two parent solutions exchange the part of solution string from that point to the end. This process produces two offspring solutions. An example with GAP5_1 solution strings is given below.

  p_1 : 7 4 3 2 6 1   3 6 8 4 2 2   1 | 5 4 8 7 7   1 4 2 5 3 6

  p_2 : 2 2 2 2 4 5   2 2 4 4 8 8   8 | 8 8 8 8 8   3 3 3 3 3 3

  Resulting offspring:

  c_1: 7 4 3 2 6 1   3 6 8 4 2 2   1 | 8 8 8 8 8   3 3 3 3 3 3

  c_2: 2 2 2 2 4 5   2 2 4 4 8 8   8 | 5 4 8 7 7   1 4 2 5 3 6

- Uniform Random Mutation: for the uniform random mutation, every number in the solution string could change with a given probability. An example with GAP5_1 solution string is given below:

  p: 7 4 3 2 6 1   3 6 $\underline{8}$ 4 2 2   1 | 5 4 8 7 7   1 $\underline{4\ 2}$ 5 3 6

  After 3-point mutation:

  p: 7 4 3 2 6 1   3 6 $\underline{5}$ 4 2 2   1 | 5 4 8 7 7   1 $\underline{7\ 4}$ 5 3 6

- Tournament-2 Selection: two individuals are randomly selected from the population. Their fitness scores are then compared. The individual with higher fitness score wins

```
• Randomly initialize a population of solutions, P, with population size N.
• GenerationCount = 0.
• while (GenerationCount < maxGenerations):
    – GenerationCount = GenerationCount + 1;
    – P' ⟵ [ ]
    – Evaluate the fitness of every individual in the current population P:
      Evaluate the individual's feasibility first.
        ∗ If individual is feasible:
          Fitness score = objective value.
        ∗ If individual is infeasible:
          Fitness score = negative Euclidian distance to the boundary of feasible region.
    – Repeat step 1-4 until P' reaches size $\frac{N}{2}$:
        1. Select two individuals from feasible group of P based on some fitness criterion.
        2. The selected two individuals mate and produce offspring.
        3. The offspring mutate with a certain probability.
        4. Add the two newly generated individuals (the offspring) to P'.
    – Repeat step 5-8 until P' reaches size N
        5. Select two individuals from infeasible group of P based on some fitness criterion.
        6. The selected two individuals mate and produce offspring.
        7. The offspring mutate with a certain probability.
        8. Add the two newly generated individuals (the offspring) to P'.
    – P = P'.
```

Figure 3: Pseudocode for the FI-2Pop GA

and become the mating candidate.

- The Composition of New Generation: all individuals from the original population are discarded after enough number of offspring individuals is reproduced. The new population consists of only the newly generated solutions.

1.4.4. The GA Parameter Setting

For both the Penalty GA and the FI-2Pop GA, the common GA parameters—population size, crossover and mutation rates—are selected via factorial experiments. Population sizes of 50 and other sizes ranging from 100 to 1000 with an increment of 100 were tested to find the most efficient value in terms of reaching a desirable level of average maximum objective value. The combination of crossover and mutation rates is chosen via factorial experiments. Possible combinations among crossover rates 0.4, 0.5, 0.6, 0.7, 0.8, and 0.9 together with mutation rates 0.01, 0.03, 0.05, 0.07, and 0.09 are investigated. Individual rate combination is tested with 5000 generation in each run. Fifteen independent runs are performed for each test. The crossover-mutation rate combination resulting in the highest average best objective value and lowest variance of the best objective value is identified for Penalty GA and FI-2Pop GA respectively.

After tuning, we set the crossover rate at 0.7 and the mutation rate at 0.03 for the FI-2Pop GA; for the Penalty GA, we used 0.7 and 0.01 respectively. Population size was 500, run length was 5,000 generations, and each problem is tested with 12 replications. For each problem, a set of solutions are collected from the last 500 of 5,000 generations. The code is written in MATLAB. In the Penalty GA, the penalty assessed for an infeasible solution was 3 $\times$ the sum of the constraint violations.

1.5. Comparison Between Penalty GA and FI-2Pop GA Results

For assessing the performance of the GAs, we answer the following five questions with the solution collections generated by Penalty GA and FI-2Pop GA, respectively. In addition

	Feasible Solutions	Infeasible Solutions
Single Constraint	A	B
Constraints Collectively	C	D

Table 2: Categories of GA Performance Assessment

to examining the best feasible solution provided by both GAs, we examine the collected solutions for the following four categories of questions (Table 2):

- Question Type A: With regard to each constraint, find the count and quality of feasible solutions with objective values close to the best known solution. To investigate the impact on objective value brought by varying single constraint, the constraints are tightened one-at-a-time by 5%. On quality, we look at feasible solutions with 99%, 99.5%, and 99.6% of the optimal value. The number of solutions that meet the target optimal value levels are tallied.

- Question Type B: With regard to each constraint, find the count and quality of infeasible solutions near to feasibility. To investigate the impact on objective value brought by varying single constraint, the constraints are relaxed one-at-a-time by 5%. The impact on objective value brought by relaxing constraints one-at-a-time by 10% is also investigated.

- Question Type C: Find the count and quality of feasible solution with objective values close to the best known solution. We examine the number of feasible solutions with an objective value greater than or equal to 99.5%, 99.6%, and 99.7% of the known optimal values.

- Question Type D: Find the count and quality of infeasible solutions with objective function equal to better than the best known value.

As a sample of the data we collect with regard to the four types of questions, Table 3 reports on what we call *feasible solutions of interest* (FoIs): feasible solutions whose objective values are near to that of the best known solution, x^+, which in this case is the optimal solution.

	$\geq 99\% z^*$		$\geq 99.5\% z^*$		$\geq 99.6\% z^*$		$\geq 99.7\% z^*$	
Problem	2Pop	Penalty	2Pop	Penalty	2Pop	Penalty	2Pop	Penalty
GAP11_1	224	652	0	53	0	25	0	11
GAP11_2	538	952	29	40	21	10	7	5
GAP11_3	89843	62582	13824	9940	6525	5185	2200	2017
GAP11_4	530	294	19	4	7	0	2	0
GAP11_5	351	332	1	1	1	0	1	0
GAP12_1	2564	7280	192	285	37	14	8	0
GAP12_2	1805	4497	74	260	16	63	5	21
GAP12_3	2009	7271	321	169	127	30	66	4
GAP12_4	15834	30811	1540	362	586	3	309	0
GAP12_5	2593	3741	146	67	36	2	12	0

Table 3: GAP11 & GAP12 FoIs: Count and quality of feasible solutions with objective function values near the best known solutions

Comparing the number of found solutions with objective values of 99% or higher than the known optimum, both GAs find many solutions for this difficult problem, but the Penalty GA outperforms the FI-2Pop GA at the 99% level. Going beyond 99%, however, the FI-2Pop GA generally outperforms the Penalty GA. Looking on the infeasible side, Table 4 reports on the infeasible solutions of interest (IoIs): infeasible solutions whose objective values are equal to or better than x^+. We see that the FI-2Pop GA is dominant, except on GAP11_1 and perhaps on GAP12_2.

From our preliminary experiments for comparing the Penalty GA and FI-2Pop GA, the followings are our observations:

- For 18 of the 21 test problems, both GAs provide the best feasible solution with objective value reaching 99.5% (or higher) of the known problem optimal values. Although it is not the goal of this study to provide optimal solutions with GAs, it gives a sense about FI-2Pop GA and Penalty GAs performance with regard to finding the optimal solutions.

- For type A questions, with regard to each constraint, the impact on objective value brought by tightening single constraint: as the target objective value goes higher, the

Problem	z^*	$\geq z^*$ 2Pop	$\geq z^*$ Penalty	$> z^*$ 2Pop	$> z^*$ Penalty
GAP11_1	1139	13	133	3	82
GAP11_2	1178	182	100	140	35
GAP11_3	1195	1039	829	204	194
GAP11_4	1171	202	24	201	18
GAP11_5	1171	20	4	11	1
GAP12_1	1451	8	0	1	0
GAP12_2	1449	3	5	0	0
GAP12_3	1433	37	2	3	0
GAP12_4	1447	21	0	5	0
GAP12_5	1446	131	0	41	0

Table 4: GAP11 & GAP12 IoIs: Count of infeasible solutions with objective function values $\geq$ or $>$ the known optimal values

FI-2Pop GA performs much better than the penalty GA. Based on a Wilcoxon signed rank test with continuity correction, the p-value = 0.0001437 for the target objective value at 99.6% of the known optimal value.

- For type B questions—with regard to each constraint, find the impact on objective value brought by varying single constraint, the constraints are relaxed one-at-a-time by X%. Penalty GA perform better than the FI-2Pop GA at the 5% relaxation level. But as the relaxation level rises, the difference is less clear between the two GAs.

- For type C questions, find the count and quality of feasible solution with objective values close to the best known solution, we examine the number of feasible solutions with an objective value greater than or equal to 99.5%, 99.6%, and 99.7% of the known optimal values. We found that as the objective value rises, FI-2Pop GA performs much better (although it has a hole with problem GAP9_1 and GAP12_12).

- For type D questions, find the count and quality of infeasible solutions with objective function equal to better than the best known value, the results show that the number of infeasible solutions provided by both GAs dwindles as the problem size grows. Interestingly, when dealing with the bigger sized problem set GAP12 (with 10 agents and 60 tasks), the Penalty GA provides almost nothing while FI-2Pop GA still offers plenty of information.

As we expected, as a meliorizing population-based metaheuristic, a GA tends to produce many solutions with similarly high fitness values (providing of course that they exist and can be found). It is just these good but non-optimal solutions that, we observe, constitute the FoIs. What about the infeasible side and the IoIs? Here we have to be concerned that standard penalty function approaches to handling infeasible solutions will not very comprehensively explore the infeasible region(s) near the feasible-infeasible boundary (or boundaries). In the extreme case, amounting to a 'death penalty' for infeasible solutions, there will be comparatively few solutions found and they will not be parents for subsequent explorations. This worry has received some empirical confirmation (Kimbrough et al. [2009b]). Based on our observations, we continue our explorations using the feasible-infeasible 2-population (FI-2Pop) GA for the rest of the experiments.

1.6. Two Different Infeasible Solution Fitness Measures

With the two-population GA, we take the objective value as the measure of a feasible solution's fitness score because we want to focus our search in the area(s) around the optimal solution(s). For the measure of the infeasible solution's fitness, there are several candidates which interest us, e.g., the sum of total constraint violations for the given infeasible solution, the maximum amount of violation among all constraint violations for the given infeasible solution, the Euclidean distance to the feasibility boundary of the given infeasible solution, and so on. The Euclidean distance comes as our first choice, since conceptually one would like to get infeasible solutions which are 'close' to the feasibility boundary. On second thought, it could also be interesting if one tries to minimize the largest distance on any single dimension between the specified solution and the feasibility boundary. To compare the effect of different fitness measures for the infeasible population, we solve five sets of GAPs with both fitness measures—the Euclidean distance and the maximum violation.

Given $I = \{1, ..., m\}$ index the set of m agents, s_i presents the slack for $agent_i$, and ns_i presents the negative slack, we consider two different measurements for the fitness of infeasible solution during the feasible-infeasible two-population GA searching:

- Euclidean Distance: to take the 'ordinary' distance between an infeasible solution and the feasibility boundary as its fitness value.

$$fitness_{E.Dist} = -\sqrt{\sum_{i=1}^{m} ns_i^2}; \qquad ns_i = \begin{cases} 0 & \text{if } s_i \geq 0 \\ s_i & \text{if } s_i < 0 \end{cases}$$

- Max Violation: to take the maximum violation of all constraints of an infeasible solution as its fitness value.

$$fitness_{Max.V} = \min_{i \in I} s_i$$

As an example, we report part of the findings with regard to the FoI counts in Table 5. We set the two-population GA with following parameter values: population size=500, number of generations=5000, trials=12, xover=0.7, mutation=0.03, heapsize=1000, feasible-threshold (feasTHD)=0, infeasible-threshold (infeasTHD)=5. No seeding for the randomly initialized population.

Table 5 shows the comparison of feasible solution of interest (FOI) counts for the two test sets with largest problem sizes (GAP 11 & 12). In terms of FOI counts, when requiring the objective value to reach at least 99.5% of the known optimal (say 99.5%, 99.6%, and 99.7%), neither measure clearly dominates the other. But in terms of the best found objective value, the Euclidean distance measure does slightly better (5 cases better, 2 cases worse, 3 cases tie). Test results for the other smaller problem size GAP sets are consistent with the one shown in Table 5 in the sense that when looking at the result problem set by problem set, neither measure clearly dominates the other for FOI counts of objective values at 99.5%-99.7% level. For the IOI counting, the solutions are counted based on wether their Euclidean

GAPID	FOI Counts								Best Found Obj.Val.	
	$\geq 99\% z^*$		$\geq 99.5\% z^*$		$\geq 99.6\% z^*$		$\geq 99.7\% z^*$			
	E.	Max	E.	Max	E.	Max	E.	Max	E.	Max
11_1	405	807	3	3	2	1	1	0	1136	1135
11_2	918	1000	49	17	16	3	3	3	1176	1176
11_3	1000	385	130	2	50	0	7	0	1192	1190
11_4	348	665	8	22	1	4	1	1	1169	1168
11_5	843	1000	3	36	1	15	0	3	1167	1168
12_1	1000	1000	874	556	175	57	43	7	1448	1447
12_2	1000	1000	799	898	112	188	32	48	1447	1447
12_3	1000	1000	120	224	1	23	0	7	1428	1430
12_4	1000	1000	1000	47	347	0	158	0	1444	1441
12_5	1000	1000	1000	1000	268	424	59	168	1444	1444
	Case Counts									
	E.	Max	E.	Max	E.	Max	E.	Max	E.	Max
Better	1	4	4	4	5	5	4	4	5	2
Tie	5		2		0		2		3	

Table 5: Comparison for two infeasible solution fitness measures. GAP11_12 SoIs count and quality of feasible solutions with objective function values near the best known solutions

distance to the feasibility boundary are within a certain given range. The overall test result (including test problem sets GAP 5, 9, 10, 11, and 12) indicates a weak trend that Euclidean distance measure does slightly better when the searched solutions are required to be very close to the feasibility boundary (either FoI or IoI). We use the Euclidean distance as the infeasible solution fitness measure for the rest of our experiments. In any case, we see potential improvement on the quality of provided information by collecting SoIs obtained with both measures. It is definitely worth further investigation with regard to different type of optimization problems.

1.7. Collecting the Solutions of Interest

Having given the reason on using FI-2Pop GA to search the potential solutions, we now illustrate the procedure of how we collect the solutions of interest (SoIs).

In terms of data structures, we use multiple heaps to record the encountered SoIs. A heap is basically a priority queue which can hold a pre-defined maximum number of solutions. Let's call the process of solving a given problem a 'run'. In a single run, the problem to be solved is fixed per our definition of a 'run', e.g., a particular GAP, and we repeat the GA searching process multiple times. Each individual GA searching process, defined as a 'trial', starts with a new randomly initialized population. With each randomized initialization, the population evolve through the pre-defined number of generations and complete a trial. A 'run' consists with a pre-defined number of trials.

A heap is set to be empty at the beginning of a run and is maintained throughout the run. At the end of a run, it contains the best solutions found over all trials in the run, by the specific criteria attached with the heap. Multiple heaps can be used in a run according to the number of types of SoIs the decision maker is interested in. Since the SoIs collecting is basically a solution logging process, it does not affect the actual search process of the GA. The solution collecting process is outlined as the pseudocode in Figure 4.

In our experiments, we set up four separate heaps to collect the four types of SoIs we care about. The four heaps are described as follows and categorized in Table 6.

	Feasible Solutions of Interest (FoIs)	Infeasible Solutions of Interest (IoIs)
Simple Selections	FoI(Obj)	IoI(SumV)
Conditional Selection	FoI(Slacks | MinObj)	IoI(Obj | MaxDist)

Table 6: Categories of SoI Heaps

All heaps come with the size parameter, MaxHeapSize, which sets the maximum number of solutions a heap can hold. We set the MaxHeapSize to 2000 solutions for our benchmark GAP experiments. Each heap holds a set of SoIs, which belong to the same category. The detailed description of our four heap categories are as follows:

- Heap FoI(Obj): records the best feasible solutions, with objective function value as the evaluation criterion.

- Heap FoI(Slacks | MinObj): records the best feasible solutions whose objective function values are no less than the specified MinObj value. The evaluation criterion is the sum of the slacks in the constraints. The MinObj is normally set at 997.5% of z^+.

```
Procedure for Collecting SoIs:

  1. Specify HashAttribute, Condition Attribute and MaxHeapSize
  2. Specify CandidateSolutions
  3. Initialize Heap. Fill up the Heap with MaxHeapSize fake solutions with poor scores
     on HashAttribute.
  4. Heap = UpdateHeap(Heap, CandidateSolutions, HashAttribute, ConditionAttribute)

==================================================

Function: UpdateHeap(Heap, CandidateSolutions, HashAttribute, ConditionAttribute)
{
  • While (CandidateSolutions ≠ [ ])
      – CurrentCandidate = CandidateSolutions(1)
      – CandidateSolutions = CandidateSolutions(2:end)
      – If (CurrentCandidate satisfies ConditionAttribute) and
        (CurrentCandidate ∉ Heap) and
        (HashAttribute of CurrentCandidate ≻
        HashAttribute of the Heap Solution which has the worst HashAttribute value)
        then
          ∗ Delete the Heap Solution with worst HashAttribute Value
          ∗ Insert Current Candidate into Heap
  • Return Heap
}
```

Figure 4: Pseudocode for Collecting Solutions

Recalling Figure 1.1 formula (3), the sum of the slacks for any given feasible solution is $\sum_{i \in I}(b_i - \sum_{j \in J} a_{ij}x_{ij})$.

- Heap IoI(SumV): records the best infeasible solutions with the sum of constraint violations as evaluation criterion. With the 'best' infeasible, we mean the infeasible solutions that have lowest sum of violations, which indicate the solutions that are closest to feasibility. Recalling Figure 1.1 formula (3) the sum of the constraint violations for any given infeasible solution is $\sum_{i \in I} min\{0, (b_i - \sum_{j \in J} a_{ij}x_{ij})\}$.

- Heap IoI(Obj | MaxDist): records the best infeasible solutions with objective function value as evaluation criterion, provided that their sum of constraint violations is no greater than the given value MaxDist. The value MaxDist is typically set at 5 for our benchmark GAPs. With the 'best' infeasible, we mean the infeasible solutions that have the highest objective function value and are near the feasible region (within the given distance MaxDist).

Other search-related information with regard to each recorded solution is also logged in the heap. Such information includes:

- feTrial: the first encounter trial ID-records the ID of the first trial which the corresponding solution is first found and logged on the heap.

- feGeneration: the first encounter generation ID-records the ID of the first generation which the corresponding solution is first found and logged on the heap in the specified trial.

With the collected SoIs at hand, many questions arise. For example:

- How can we determine whether we have collected 'enough' SoIs or reasonably so?

- Given that after the pre-defined number of trials, should our heaps be not filled with MaxHeapSize SoIs, or should we settle for fewer SoIs? Since we could still use the information at hand to identify areas of interest, will we be better off to follow up with focused efforts to further search the specific area?

- Are some SoIs typically harder to find than others?

We may be able to get some insight with the help of the search related information which we collected. Take our GAP5_1 FoI(Obj) heap as an example. Figure 5 shows the number of new entries in the FoI(Obj) heap and IoI(Obj|MaxDist) heap during each of the 200 trials. For the IoI(Obj|MaxDist) heap, there is a trend of dwindling on the number of new entries as the search approaching the end of 200 trials; while it is not so apparent on the diminishing number of new entries on the FoI(Obj) heap searching process. To get a sense of whether we have collected 'enough' SoIs, a little modeling is helpful.

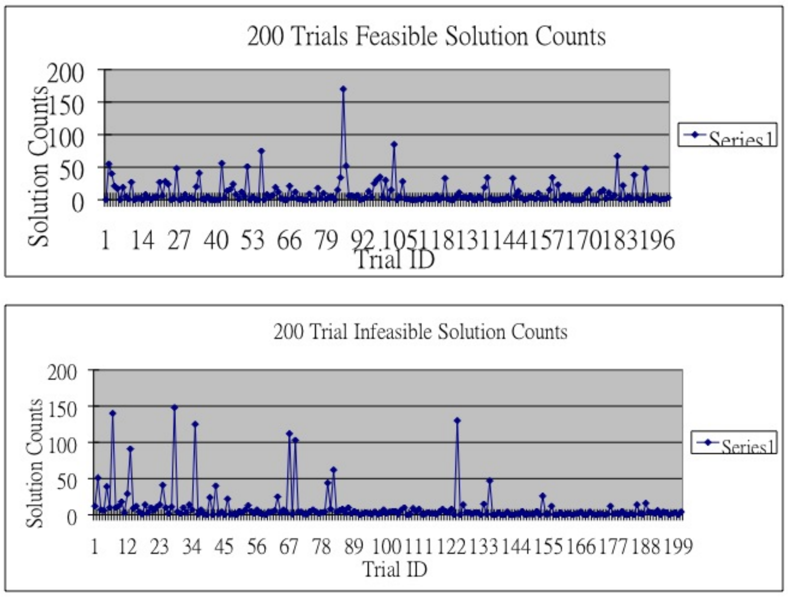


Figure 5: GAP5-1 FoIs and IoIs Counts, 200 Trials

Let us say that a heap is *complete* if it is filled with MaxHeapSize SoIs with the best evaluation values provided that they satisfy the given condition(s). Since our SoI collecting procedure never deletes any SoI that is recorded before the heap is entirely filled, we can thus expect that over multiple trials, the heap will move towards completion. Let us assume that it takes equal amount of effort to find each of the solutions collected in the heap, and assume that MaxHeapSize is 1000, our GA on average finds 10 of the 1000 SoIs per trial. The probability that a given SoI is not found in n independent trials is $(1-0.01)^n = 0.99^n$. If we are expecting to have the heap with 90% completion, then we need $1 - 0.99^n = 0.9$ or $n = ln0.1/ln0.99 \rightsquigarrow n > 229$. In other words, suppose a solution of interest is only be found in 1 in 100 trials, and we want a 90% chance of finding it. Then we need to have at least 229 trials of randomly initialized searching process.

The results reported here with respect to GAP5-1 is similar to what we have found with other problems in the Beasley GAP test suite (Beasley [2009]). Broadly speaking, there are a surprisingly large number of SoIs (both feasible and infeasible). This implies important opportunities that are not currently well exploited for supporting post-solution deliberation.

1.8. Using the Solutions of Interest

To test our approach, we build our deliberation support system that searches the solutions of interest with FI-2Pop GA and gathers four types of SoIs with prioritized solution collecting procedure. Extensive runs are carried out under various settings, on the Beasley OR-Library GAP sets (Beasley [2009]). In this section, we report our findings for the GAP set 4 problem 2 (c530-2) as a representative example of our experiments. GAP4-2 is a problem with 5 machines, 30 jobs, and its known optimal objective value is 644. The main purpose of this section is to demonstrate how having the solutions of interest at hand may actually contribute to improved decision-making. We will illustrate why the SoIs could be interesting by examining five types of solutions that interest us: (1) the solution(s) solving the given problem to optimality, (2) the feasible solutions which are near-optimal, (3) the feasible solutions that have objective value no worse than a given threshold and spare more units of certain resources, (4) the infeasible solutions that only require minimum amount of extra resources to become feasible, and (5) the infeasible solutions that have high objective function value but are not overly infeasible (within a tolerable distance to the feasible boundary).

	0	1	2	3	4	5	6	7	8	9
0	-	3	3	5	1	2	1	4	1	4
1	2	3	2	1	4	4	5	2	2	5
2	3	4	5	3	5	3	1	4	1	5
3	2	-	-	-	-	-	-	-	-	-

(a) Optimal Solution # 1

	0	1	2	3	4	5	6	7	8	9
0	-	3	3	5	1	2	1	3	1	4
1	2	3	2	1	4	5	5	2	4	5
2	3	4	5	3	4	2	1	4	1	5
3	2	-	-	-	-	-	-	-	-	-

(b) Optimal Solution # 2

Table 7: Two Optimal Solutions for GAP4-2

1.8.1. Optimal Solutions

Using the deliberation system, we found not only one but two solutions with the known optimal objective value of 644. Table 7 shows the two optimal solutions found. The differences are shown in red (assignments for job 7, 15, 18, 24, and 25). We note that (i) conventional solvers will typically find just one optimal solution and (ii) there is generally real value in knowing more than one. For example, one optimal solution to the *model* may be preferable to another because the underlying problem has changed or has relevant aspects that are not captured in the model.

1.8.2. Near-Optimal Feasible Solutions

Given that the known optimal objective value is 644, our deliberation support system provides eight other feasible solutions with objective value 643. Although there is one unit short on the objective value, these eight solutions spare more units of certain resources. To compare the slacks on resources among the optimal/near-optimal solutions, we display them in terms of resource usage in Table 8. Should a resource be able to be used in other more profitable ways (e.g., resource 4, R4, which corresponds to the constraint whose RHS value is b_4), the decision-maker may be interested in other alternative solution (e.g., solution number 3, which spares 11 units of resource 4). The sampling results also show that we have fewer opportunities for redeploying resource 3 and resource 5. Since solution #6 spares 5 units of resource 5, it may be a tempting solution should the decision-maker consider the spared resource 5 worth more than the 0.002% reduction on objective value. Solution #8 with objective value 643 may be preferred in a stochastic world, since it has slack of at least

1 unit on every resource while neither of the optimal solutions have this property.

Since the sampled SoIs in the heaps can be ranked according to either their objective values, the sum of slacks, or the slack on a certain resource, it is helpful when the decision-maker tries to answer the what-if question involving changing availability for one or more resources. For example, both of the current optimal solutions have slack 2 on resource 1. If for any reason, one needs solution which can spare at least 5 units of resource 1, solution #1 and #6 will both meet such requirement with a reduction of less than 0.002% on objective value from the optimality. By carefully examining the near-optimal solutions, we also find relevant information on other types of deliberation question. For example, Table 7 shows that, in the two optimal solutions, job 25 is assigned to machine 3 and machine 2 respectively. Why not assign task 25 to machine 1? What will happen if job 25 is assigned to machine 1? The near-optimal solution #4 listed in Table 8 actually has task 25 assigned to machine 1, with an objective value 643. We conclude that if task 25 is assigned to machine 1, the best solution we have is with objective value 643. Table 9 details the task assignments of solution #4.

1.8.3. High Slacks Feasible Solutions

The second type of FoIs are the feasible solutions with high slack sum given that their objective values are higher than a specific threshold. Table 10 displays the sampled solutions with high slack sum ranked according to their objective values. Depending on the actual economic opportunities, solution #1, 2, and 3, which have slack sums no less than 36, with their objective value above 97.6% of the optimal value 644, may provide significant savings on resources from either of the optimal solutions and might be a good bargain as well.

1.8.4. Nearly-Feasible Solutions

To improve the objective value beyond the current optimal value, the decision maker may be interested in identifying the achievable improvement with minimal additional resources, given that extra resources are hard to get. The nearly-feasible solutions of the IoIs provide

	Obj.Val.	R1	R2	R3	R4	R5
0	644	2	1	1	2	0
0	644	2	5	0	1	1
1	643	5	4	1	3	0
2	643	0	2	1	0	0
3	643	0	4	1	11	0
4	643	2	1	1	3	0
5	643	0	1	0	7	0
6	643	5	4	0	1	5
7	643	2	1	0	4	0
8	643	2	5	1	4	1
9	642	0	4	3	12	1
10	642	9	0	0	0	2
11	641	0	8	0	1	3
12	640	7	1	1	0	0
13	640	5	4	7	2	0
14	640	0	1	4	0	9
15	640	10	4	1	0	0

Table 8: GAP4-2 Optimal & Near-Optimal Feasible Solutions; from `FoI(Obj)`

	0	1	2	3	4	5	6	7	8	9
0	-	3	3	5	1	2	3	4	1	4
1	2	4	2	1	4	5	5	2	2	5
2	3	4	3	3	5	1	1	4	1	5
3	2	-	-	-	-	-	-	-	-	-

Table 9: GAP4-2: Best found solution with job 25 assigned to machine 1

such information. The solutions recorded in the `IoI(SumV)` heap are indeed the infeasible solutions which have the shortest distance to the feasible boundary. Since such search focuses on solutions with minimal constraint violation, the sampled nearly-feasible solutions tend to be one distance unit away from being feasible. Our approach with the FI-2Pop GA typically finds a rich collection of one-away infeasible solutions. Table 11 lists three infeasible solutions which are one unit short of either resource 3 or resource 4. Should the required resource is obtained, one will be able to achieve the higher new optimal objective value 648 (instead of the known optimal value 644).

	Obj.Val.	Slack Sum	R1	R2	R3	R4	R5
0	644	6	2	1	1	2	0
0	644	9	2	5	0	1	1
1	629	37	0	28	3	5	1
2	629	36	0	8	10	17	1
3	629	36	20	5	0	4	7
4	628	39	15	15	3	5	1
5	627	38	0	21	10	4	3
6	626	42	0	8	16	17	1
7	626	40	5	28	6	0	1
8	626	40	15	8	10	4	3
9	626	40	15	8	3	5	9
10	626	39	15	9	3	11	1
11	626	39	20	8	7	4	0
12	626	39	0	8	9	22	0
13	625	42	20	15	6	0	1
14	625	41	0	1	3	29	8

Table 10: GAP4-2 FoIs with large sums of slacks from `FoI(Slacks|MinObj)`

1.8.5. High Objective Value Infeasible Solutions

Another common deliberation focus is how to best allocate the extra resources in order to achieve the new optimal objective value, given that there is limited extra budget available. The solutions recorded in the `IoI(Obj|MaxDist)` heap are the infeasible solutions with high objective values, provided they are not too far away from being feasible. The 'not too far away' is specified during the search with a given constraint violation threshold (5 distance unit in our example). Table 12 shows such solutions found by our deliberation system, ranked on the expected objective values.

1.9. Extended Experiments on the Generalized Quadratic Assignment Problem

Since the generalized quadratic assignment problem remains very difficult for CPLEX and other standard OR solvers, it is an attractive class of problem to test our metaheuristic approach. To further demonstrate the effectiveness of our proposed approach, we apply the Deliberation Decision Support System (DDSS)[3] on three sets of generalized quadratic

[3]DDSS is our application developed using MATLAB, based on the proposed approach.

	Distance to Boundary×-1	Objective Value	R1	R2	R3	R4	R5
1	-1	648	5	4	0	-1	0
2	-1	648	2	1	0	-1	0
3	-1	648	0	4	-1	0	1
4	-1	645	8	1	1	-1	0
5	-1	645	0	4	1	-1	5
6	-1	644	0	4	10	-1	0
7	-1	644	0	2	4	-1	0
8	-1	643	5	-1	0	0	0
9	-1	640	-1	9	0	9	0
10	-1	639	0	8	-1	10	1
11	-1	638	5	-1	0	11	3
12	-1	637	5	1	0	16	-1
13	-1	637	16	1	0	2	-1
14	-1	636	-1	7	4	1	0
15	-1	635	5	18	1	-1	1
16	-1	635	5	8	14	-1	1

Table 11: GAP4-2 IoIs with min constraint violations from `IoI(SumV)`

assignment problems (GQAP). The first two sets of problems consist of three different sizes of bench mark problems from Elloumi's web site. The third problem set contains one medium size crossdock assignment problem. We will describe the Elloumi's benchmark problems first and discuss the details about crossdock problem in the later part of this section (1.9.5).

We select twenty four problem instances with known optimal values from Elloumi's web site dedicated to the Constrained Task Assignment Problem (CTAP, Elloumi [2014]). Coming under the class of generalized quadratic assignment problem, Elloumi's CTAP instances are all about the module allocation problem (MAP of distributed computing systems design for VLSI). In the CTAP, a set of program modules (or tasks) is to be executed by a set of processors, subject to both execution and inter-module communication costs. Assuming that each processor has limited memory and the communication links are identical (the communication costs of pairs of modules are said to be *uniform* since they are independent from the processors and they are symmetric). The goal is to find a suitable assignment of program modules to processors which minimizes the overall cost without exceeding any processor's memory capacity limit.

To define the problem, we use the following notation:

- M: a set of program modules. Let $I,\ J = \{1, ..., m\}$ index the set of modules.
- N: a set of processors. Let $H,\ K = \{1, ..., n\}$ index the set of processors.
- x_{ik}: the decision variable x_{ik} is set to 1 if module i is assigned to processor k, 0 otherwise.
- r_i: the memory requirement of module i.
- q_k: the limited memory of processor k.
- e_{ik}: the execution cost of module i on processor k.
- c_{ij}: the communication cost occurring when modules i and j are assigned to different processors; $c_{ij} = c_{ji}$.

	Distance to Boundary×-1	Objective Value	R1	R2	R3	R4	R5
1	-4.1231	652	-2	4	-2	-3	0
2	-2.8284	651	-2	4	0	-2	1
3	-2.8284	651	-2	-2	0	5	2
4	-5	650	-0	4	0	1	-5
5	-5	650	2	-5	1	0	1
6	-2.8284	650	-2	4	0	-2	1
7	-3.4641	650	-2	4	0	-2	-2
8	-3.6056	650	-3	-2	0	1	0
9	-4.2426	650	-3	1	1	-3	0

Table 12: GAP4-2 IoIs Ranked by objective value

The CMAP is formulated as follows:

$$\min_{x_{ik}} \quad \sum_{i=1}^{m-1}\sum_{j=i+1}^{m} c_{ik} + \sum_{i=1}^{m}\sum_{k=1}^{n} e_{ik}x_{ik} - \sum_{i=1}^{m-1}\sum_{j=i+1}^{m}\sum_{k=1}^{n} c_{ij}x_{ik}x_{jk} \tag{1.8}$$

$$\textit{subject to}: \quad \sum_{k\in N} x_{ik} = 1 \qquad \forall\, i \in M, \tag{1.9}$$

$$\sum_{i\in M} r_i x_{ik} \le q_k \qquad \forall\, k \in N, \tag{1.10}$$

$$x_{ik} \in \{0,1\} \qquad \forall i \in M,\ \forall k \in N. \tag{1.11}$$

The constraints, including the integrality condition on the variables, state that each program module is assigned to exactly one processor, and that the memory capacity of each processor is not exceeded.

Matrices **E**, **C**, and vectors **R** and **Q** with elements e_{ik}, c_{ij}, r_i, and q_k are the parameters of the model. Matrix **X** with element x_{ik} is the decision variable.

A total of eight types of Elloumi's CTAP instances are generated with four different configurations. For each configuration, two classes of instances are generated: a class with a complete communication cost matrix **C** (instance naming started with 'c'), and a second class where the non-zero elements of **C** is of 50%. The instance configurations are listed in Table 13. Our twenty four selected instances cover six of the eight provided types (i.e. type cA, A, cB, cC ,C, and cD). Since Elloumi's web site provides optimum values only for the 20 smaller cases (10×3), we obtain the optimum values of the other four larger cases (20×5 and 24×8) from Hahn et al. [2008].

	Configuration A	Configuration B	Configuration C	Configuration D
execution cost	[0,100]	[0,10]	[0,100]	0
communication cost	[0,100]	[0,100]	[0,10]	[0,100]

Table 13: GQAP: Elloumi Instance Configurations

1.9.1. GA Solution Representation

Following our GAP solution convention, solutions for CTAP are also represented as a string of integers. The i^{th} number in the solution string indicates the ID of a processor which is assigned to perform the i^{th} program module. To illustrate this presentation, consider the scenario which has twenty-four program modules and eight processors. The processors have identifier numbered from 1 to 8. There are twenty-four integers on the solution string, each integer is in the range of [1..8]. A solution string may look like following:

$$7\,4\,3\,2\,6\,1 \quad 3\,6\,8\,4\,2 \quad 1\,6\,4\,8\,7\,7 \quad 1\,4\,2\,3\,6$$

1.9.2. The Local-Path Crossover Operator

During preliminary GQAP benchmark testing (and consistent with prior studies), we find that path crossover (originally developed by Glover [1994]) works better than the regular crossover and the order crossover in terms of finding solutions which achieve close to known optimal objective values. However, the resulting solutions are not satisfactory, since the found FoIs are high valued suboptimal for the minimization problems and the found IoIs are still quite far away from the feasibility boundaries.

Unlike prior related studies which tried to improve GAs to solve GQAP, we take neither the route of adding periodic local optimization (Ahuja et al. [2000]) nor the route adding post-optimization Tabu search (Cordeau et al. [2006]). Our approach, however, is to keep the two population GA as simple as possible and aiming on improving its effectiveness by changing the genetic crossover operator since it is the most crucial part of a successful genetic algorithm.

Inspired by Drezner's 'merging process' (Drezner [2003]) and Glover's 'path relink' (Glover and Laguna [2000]), we developed a new crossover operator and call it the 'local path crossover'. The 'local path crossover' consists of three processing phases. These phases are described in detail as follows:

Given that there are m tasks need to be processed by n processors, our solution representation is a chromosome of m integers $x_1 x_2 \cdots x_m,\ x_i \in [1 \cdots m]$. The value of x_i indicates the processor's ID which task i is assigned to.

- Identifying the Common Genes: Let the chromosomes of parent1, parent2, and their children be represented by $x_1\ x_2\ x_3 \cdots x_n$, $y_1\ y_2\ y_3 \cdots y_n$, and $z_1\ z_2\ z_3 \cdots z_n$. Similar to path relink, the children inherit any genes common to both parents; that is, if $x_c = y_c$ for some $c \in \{1 \cdots n\}$, then $z_c = x_c = y_c$.

- Nudging: Starting from left to right, we examine the chromosomes corresponding to parent1 and parent2. If the alleles at the current position being looked at are the same , we move to the next position; otherwise, we try to make both parents be one more gene alike to their partners respectively. In order to bring one parent to be one-gene closer to the other, we take the following steps:

  For $k = 1$ to m:

  1. Assuming that the current position being looked at are position k, with the k^{th} task's assignment value (x_k).

  2. Taking parent1 ($x_1\ x_2\ x_3 \cdots x_n$) as the example solution, parent2 as the base solution. Duplicate the chromosome of the base solution (parent2), let's call the new copy *temp*. (*temp* $= z_1\ z_2\ z_3 \cdots z_n$).

  3. In *temp*, find the first position p such that $z_p = x_k$.

  4. Bring z_p to the k^{th} position of *temp* by a sequence of either wrapped right shifting or wrapped left shifting, skipping all the common genes identified in the identification process—just leave them where they are. The wrapped shifting is done with only the segment between position p and k. Such modification tries to preserve base solution (parent2) pattern in *temp* as much as possible since our

goal is just to move base solution one-gene closer to example solution (parent1).

5. Record the resulting $child_{2k}$.

6. Switch the roles of parent1 and parent2 and repeat steps 2-4; i.e., now taking parent2 as the example solution and having parent1 as the base solution. Repeat steps 2-4 with the goal of bring parent1 to be one-gene closer to parent2. Record the resulted child as $child_{1k}$.

At the end of the nudging process, we collect a set of $2m$ children—m $child_{1k}$s and m $child_{2k}s$.

Figure 6 illustrates the nudging process with wrapped right shift step by step.

- Selection: we select two solutions from the pool which combines the $2m$ children with the 2 parents base on the following rules:
  - If every solution in the pool is feasible, pick feasible solutions with the best and the worst objective values.
  - If every solution in the pool is infeasible, pick infeasible solutions with the best and the worst fitness values.
  - If the pool contains both feasible and infeasible solutions, pick the best feasible and the best infeasible solutions.

After the identification, nudging, and selection processes, two children chromosomes are generated and added into the new generation. By identification, we keep the common pattern which the two winners (of two tournament-2 selections) both possess. With nudging, we build the promising common pattern one step further at a time in a greedy manner. With the selection scheme, we keep both the best and the worst solutions in either feasible or infeasible groups to maintain the diversity of the whole population. Also, by keeping the best of both feasible and infeasible solutions, we lead the GA searching toward the boundary

Step0: Starting nudging process with k=1.

Step1: Identify the common genes.

parent 1: 5 2 $\underline{3}$ $\underline{4}$ 1 7 6
parent 2: 2 1 $\underline{3}$ $\underline{4}$ 6 5 7

Step2: Taking parent2 as the base solution,
make a copy of it as *temp*.

Step3: Find the first position in *temp*
where z_p equals $parent1(k=1) \rightarrow x_1 = 5$.
$\Rightarrow$ p=6.

parent 1: $\hat{5}$ 2 $\underline{3}$ $\underline{4}$ 1 7 6
temp: 2 1 $\underline{3}$ $\underline{4}$ 6 $\hat{5}$ 7

Step4: We care about only the segment between
position 1 and 6 & we skip the common genes.
So, do a wrapped right shift without
touching p7 -- shift every z_i one position
to the right, ignore common genes z_3, z_4 and
z_7; since z_6 is the last element, it wrapped
around and got squeezed to position 1.

Step5: Record it as $child_{21}$.

$child_{21}$: $\hat{5}$ 2 $\underline{3}$ $\underline{4}$ 1 6 7

Step6-8:Taking parent1 as the base solution, repeat
Step2-4. Now, the segment p2p1 is smaller.
Do wrapped left shift. Record it as $child_{11}$.

temp: 5 $\ddot{2}$ $\underline{3}$ $\underline{4}$ 1 7 6
parent 2: $\ddot{2}$ 1 $\underline{3}$ $\underline{4}$ 6 5 7

By the end of k=1 nudging process, we have two
children $child_{11}$ & $child_{21}$ originated from parent1 &
parent2.

parent 1: $\hat{5}$ 2 $\underline{3}$ $\underline{4}$ 1 7 6 $child_{11}$: $\ddot{2}$ 5 $\underline{3}$ $\underline{4}$ 1 7 6
parent 2: $\ddot{2}$ 1 $\underline{3}$ $\underline{4}$ 6 5 7 $child_{21}$: $\hat{5}$ 2 $\underline{3}$ $\underline{4}$ 1 6 7

Figure 6: Wrapped Right Shifting Nudging Example (when k = 1)

which divides the feasible and infeasible areas.

Our experiment results show that the local path crossover is very effective. Without the help of additional periodic local optimization nor the post-optimization searching, our two population GA with the new crossover scheme finds the known optimum solutions for the GQAP benchmark problems.

1.9.3. Elloumi c1003 Test Cases

For the twenty 10×3 'c' type instances, the 2PopGA parameters are set as follows: population size: 50, generations: 100, trials: 10, crossover rate: 0.3, and mutation rate: 0.02. Solution selection parameter settings are as follows: feasible/infeasible solution heap size: 100, feasible threshold: 0, infeasible threshold: 5. Our deliberation decision support system (DDSS) solves all twenty problems to optimality. It also finds solutions of interest (SoIs) for all twenty problems. A summary of the offered SoI counts is listed in Table 14.

On the infeasible solution of interest (IoI) side, we show the number of one-resource-away solutions, and the number of infeasible solutions which are less than or equal to three-resource-units away. Since the CMAP instances all have constraints with regard to the same resource (memory), we list the percentage increment for every additional unit memory required vs. original total capacity. On the feasible solution of interest (FoI) side, we list the number of offered solutions which have objective values no worse than 5% and 10% of the known optimal values. E.g., for instance c1003Aa, we found one infeasible solution which requires only one extra unit of memory in order to become feasible (column 2). That one unit of extra memory equals to 1.19% of the original total capacity (column 3). There are other eight infeasible solutions which are within three unit of memory shortage (column 4). As for suboptimal feasible solutions, there are 15 solutions which achieve no worse than 1.05 times of the known optimal value (column 5), and 48 other feasible solutions achieve no worse than 1.10 times of the known optimal value (column 6). The objective value improvement made with an additional unit of resource for the six instances with one-resource-away IoIs is listed in Table 15. E.g., for instance c1003Dc, the DDSS offers 18 infeasible solution of interest which are just one unit resource away from being feasible. The potential objective value improvement ranges from 2.03% to 12.03% with the eighteen offered IoIs. To be more specific, by adding an extra unit of memory to processor 3, one can achieve 12% of improvement on the objective value (see the black triangle indicator on Table 15, first row for instance 'c1003Dc').

	IoI Counts			FoI Counts	
	1-R-away	R increase %	≤ 3 -R-away	1.05*opt.val.	1.10*opt.val.
c1003Aa	1	1.19	8	15	48
c1003Ab	0	1.54	3	2	5
c1003Ac	0	1.27	0	6	42
c1003Ad	0	0.96	0	0	20
c1003Ae	0	0.93	1	0	2
c1003Ba	0	1.37	0	6	24
c1003Bb	0	1	4	7	14
c1003Bc	3	1.85	17	6	18
c1003Bd	0	1.19	2	3	15
c1003Be	0	1.04	3	1	1
c1003Ca	4	1.37	7	16	63
c1003Cb	1	1	1	10	72
c1003Cc	1	1.85	5	9	41
c1003Cd	0	1.19	0	4	20
c1003Ce	0	1.04	0	3	17
c1003Da	0	1.14	0	0	2
c1003Db	0	2	0	4	8
c1003Dc	18	1.14	41	7	34
c1003Dd	0	1.47	1	3	5
c1003De	0	0.88	7	4	22

Table 14: Elloumi c1003 Problem Set Solution of Interest (SoI) Findings

1.9.4. Larger Elloumi Test Cases

Four other larger size Elloumi CMAP test problems with known optimum values were all solved to optimality. Among the four cases, we display the findings of cases c2005De and 2408Aa since they are considered 'very hard to solve' (with solution space 9.5×10^{13} and 4.7×10^{21} respectively). See appendix Table 42 for problem c2005De configurations, Table 43 and Table 44 for problem 2408Aa configurations.

Instance	Opt.Val.	IoI Obj.Val.	Improvement	R1	R2	R3
c1003Aa	1616	1537	4.89	-1	0	21
c1003Bc	1154	1087	5.81	-1	5	1
		1121	2.86	-1	6	0
		1133	1.82	-1	6	0
c1003Ca	455	441	3.08	1	12	-1
		445	2.20	4	9	-1
		451	0.88	2	11	-1
		454	0.22	10	3	-1
c1003Cb	467	457	2.14	-1	19	24
c1003Cc	475	468	1.47	5	-1	1
c1003Dc	▶ 1230	1082	12.03	0	26	-1
		1096	10.89	10	16	-1
		1096	10.89	18	8	-1
		1099	10.65	10	16	-1
		1099	10.65	18	8	-1
		1103	10.32	0	26	-1
		1105	10.16	-1	26	0
		1106	10.08	18	8	-1
		1106	10.08	10	16	-1
		1106	10.08	20	6	-1
		1106	10.08	8	18	-1
		1144	6.99	10	16	-1
		1170	4.88	20	6	-1
		1170	4.88	8	18	-1
		1200	2.44	0	26	-1
		1204	2.11	0	26	-1
		1204	2.11	-1	26	0
		1205	2.03	0	26	-1

Table 15: Elloumi c1003 Problem Set IoI Information

- c2005De (optimal value 5435): for instance c2005De, GA parameters are set as follows: population size: 200, generation: 1000, trial: 10, crossover rate: 0.3, and mutation rate: 0.035. Solution selection parameter settings are unchanged: feasible/infeasible solution heap size: 100, feasible threshold: 0, infeasible threshold: 5.

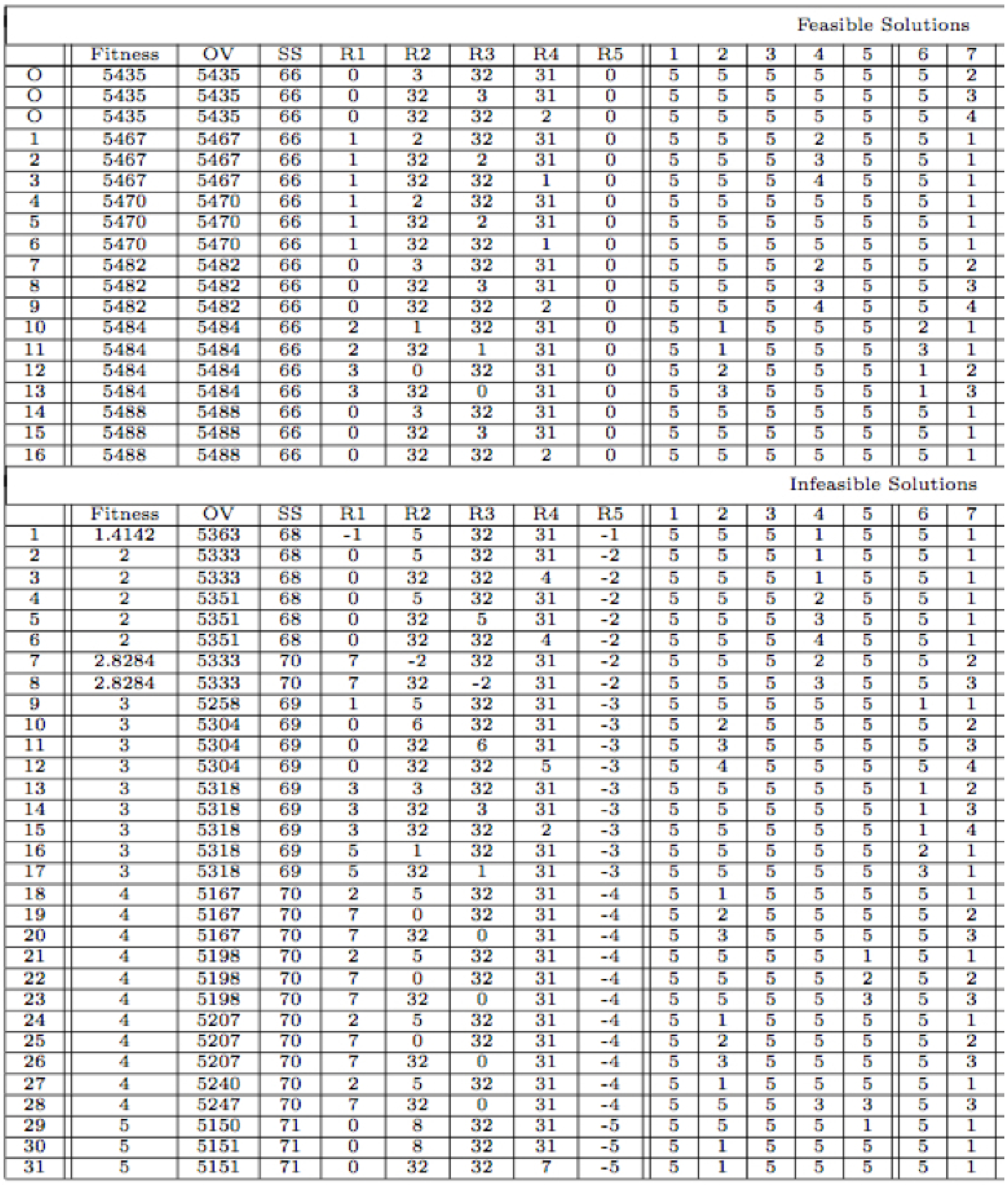
  Our deliberation support system finds all three optimal solutions and offers many other solutions of interest. In Table 7 and Table 8, we display the top SoIs according to the achieved objective value in their resource usage mode. On the IoI side, we see that, with solution #1, one can reduce 72 (1.34%) of the total cost by acquiring one extra unit of memory for processor 1 and processor 5 respectively. By obtaining two additional unit of memory for processor 5 (solution #2), one can improve the total cost by 102 points (from 5435 down to 5333, which is a 1.88% improvement). If extra 3 units of memory for processor 5 is available (solution #9), then one can further bring down the total cost to 5258 (3.26% improvement).

- 2408Aa (optimal value 5643): for instance 2408Aa, GA parameters are set as follows: population size: 200, generation: 6000, trial: 10, crossover rate: 0.3, and mutation rate: 0.04. Solution selection parameter settings are unchanged: feasible/infeasible solution heap size: 100, feasible threshold: 0, infeasible threshold: 5. Again, on top of finding the optimal solution, the DDSS offers plenty of SOIs.

  Feasible solution alternatives: Table 9 and Table 10 list ten alternative feasible solutions which achieve top objective values ranging within 0.05% - 0.27% suboptimal (obj.val. 5646-5658). Solution #2 takes a objective value deduction of 3 but it holds 11 units of slacks for resource 6. Solution #4 takes an objective value deduction of 5 but holds 7 units of slacks for resource 5. Solution #7 has an even lower objective value of 5657 but holds 8 units of slacks for resource 1. These solutions may be desirable in the cases when difference resource leftovers can be used in other more profitable ways.

  Infeasible solution alternatives: GAs offer many infeasible alternative solutions. In Table 9 and Table 10, we list part of the solution collection for demonstration. In the case of potential budget increase for one extra unit of memory, there are eighteen offered IOIs which achieve a reduction between 0.82% and 2.23% on the total cost (IoI #1-18, objective value 5597-5517). By adding one extra unit of memory for processor 4, solution #1 can reduce the total cost by more than 2%. By offering the same extra unit of memory for processor 4, solution #13 not only bring down the total by more than 1% but can also offer a high amount of memory slacks on processor 6. If we can add two extra units of memory for processor 4, then selecting solution #21can improve the total cost by more than 2.6% is achievable. By adding 2, 1, and 3 extra units of memory for processor 2, 3, and 4 respectively, solution #32 can reach objective value

5323, which is a 5.67% improvement over the optimal solution of 5643.



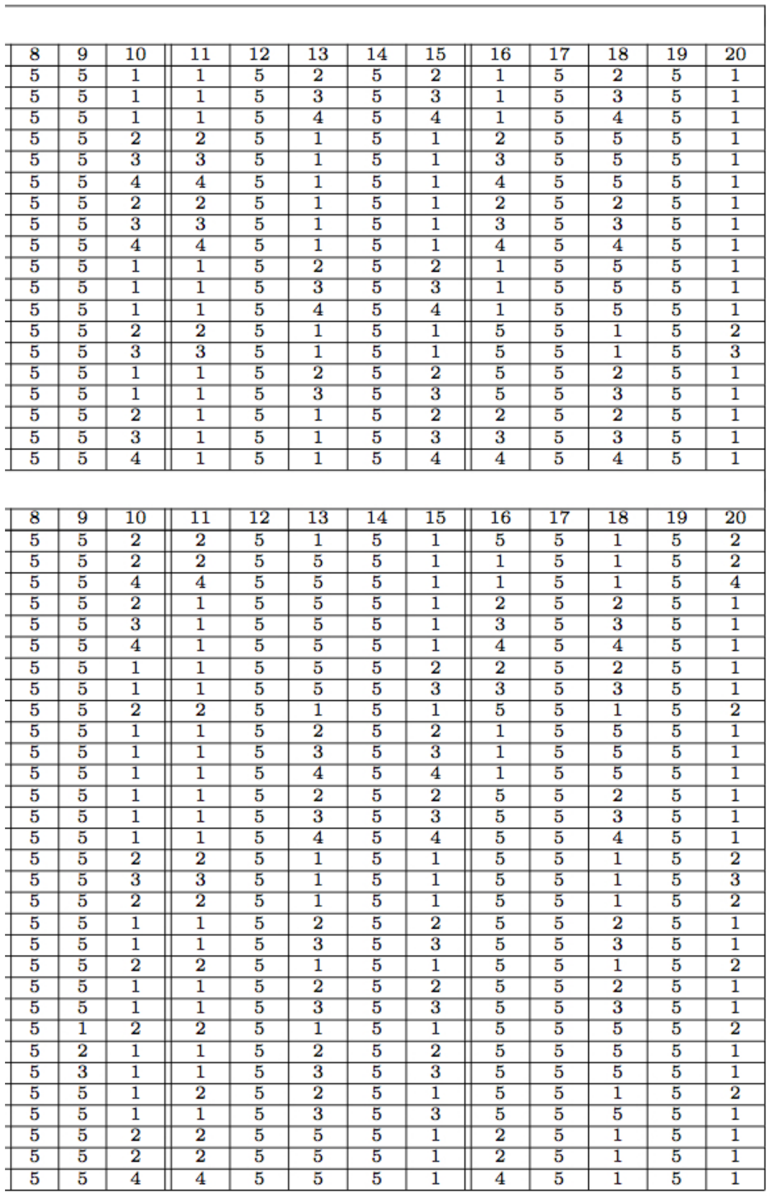
Feasible Solutions

	Fitness	OV	SS	R1	R2	R3	R4	R5	1	2	3	4	5	6	7
O	5435	5435	66	0	3	32	31	0	5	5	5	5	5	5	2
O	5435	5435	66	0	32	3	31	0	5	5	5	5	5	5	3
O	5435	5435	66	0	32	32	2	0	5	5	5	5	5	5	4
1	5467	5467	66	1	2	32	31	0	5	5	5	2	5	5	1
2	5467	5467	66	1	32	2	31	0	5	5	5	3	5	5	1
3	5467	5467	66	1	32	32	1	0	5	5	5	4	5	5	1
4	5470	5470	66	1	2	32	31	0	5	5	5	5	5	5	1
5	5470	5470	66	1	32	2	31	0	5	5	5	5	5	5	1
6	5470	5470	66	1	32	32	1	0	5	5	5	5	5	5	1
7	5482	5482	66	0	3	32	31	0	5	5	5	2	5	5	2
8	5482	5482	66	0	32	3	31	0	5	5	5	3	5	5	3
9	5482	5482	66	0	32	32	2	0	5	5	5	4	5	5	4
10	5484	5484	66	2	1	32	31	0	5	1	5	5	5	2	1
11	5484	5484	66	2	32	1	31	0	5	1	5	5	5	3	1
12	5484	5484	66	3	0	32	31	0	5	2	5	5	5	1	2
13	5484	5484	66	3	32	0	31	0	5	3	5	5	5	1	3
14	5488	5488	66	0	3	32	31	0	5	5	5	5	5	5	1
15	5488	5488	66	0	32	3	31	0	5	5	5	5	5	5	1
16	5488	5488	66	0	32	32	2	0	5	5	5	5	5	5	1

Infeasible Solutions

	Fitness	OV	SS	R1	R2	R3	R4	R5	1	2	3	4	5	6	7
▶ 1	1.4142	5363	68	-1	5	32	31	-1	5	5	5	1	5	5	1
▶ 2	2	5333	68	0	5	32	31	-2	5	5	5	1	5	5	1
3	2	5333	68	0	32	32	4	-2	5	5	5	1	5	5	1
4	2	5351	68	0	5	32	31	-2	5	5	5	2	5	5	1
5	2	5351	68	0	32	5	31	-2	5	5	5	3	5	5	1
6	2	5351	68	0	32	32	4	-2	5	5	5	4	5	5	1
7	2.8284	5333	70	7	-2	32	31	-2	5	5	5	2	5	5	2
8	2.8284	5333	70	7	32	-2	31	-2	5	5	5	3	5	5	3
▶ 9	3	5258	69	1	5	32	31	-3	5	5	5	5	5	1	1
10	3	5304	69	0	6	32	31	-3	5	2	5	5	5	5	2
11	3	5304	69	0	32	6	31	-3	5	3	5	5	5	5	3
12	3	5304	69	0	32	32	5	-3	5	4	5	5	5	5	4
13	3	5318	69	3	3	32	31	-3	5	5	5	5	5	1	2
14	3	5318	69	3	32	3	31	-3	5	5	5	5	5	1	3
15	3	5318	69	3	32	32	2	-3	5	5	5	5	5	1	4
16	3	5318	69	5	1	32	31	-3	5	5	5	5	5	2	1
17	3	5318	69	5	32	1	31	-3	5	5	5	5	5	3	1
18	4	5167	70	2	5	32	31	-4	5	1	5	5	5	5	1
19	4	5167	70	7	0	32	31	-4	5	2	5	5	5	5	2
20	4	5167	70	7	32	0	31	-4	5	3	5	5	5	5	3
21	4	5198	70	2	5	32	31	-4	5	5	5	5	1	5	1
22	4	5198	70	7	0	32	31	-4	5	5	5	5	2	5	2
23	4	5198	70	7	32	0	31	-4	5	5	5	5	3	5	3
24	4	5207	70	2	5	32	31	-4	5	1	5	5	5	5	1
25	4	5207	70	7	0	32	31	-4	5	2	5	5	5	5	2
26	4	5207	70	7	32	0	31	-4	5	3	5	5	5	5	3
27	4	5240	70	2	5	32	31	-4	5	1	5	5	5	5	1
28	4	5247	70	7	32	0	31	-4	5	5	5	3	3	5	3
29	5	5150	71	0	8	32	31	-5	5	5	5	5	1	5	1
30	5	5151	71	0	8	32	31	-5	5	1	5	5	5	5	1
31	5	5151	71	0	32	32	7	-5	5	1	5	5	5	5	1

Figure 7: Elloumi Test Problem 2005De Solutions of Interest (First Half Table)



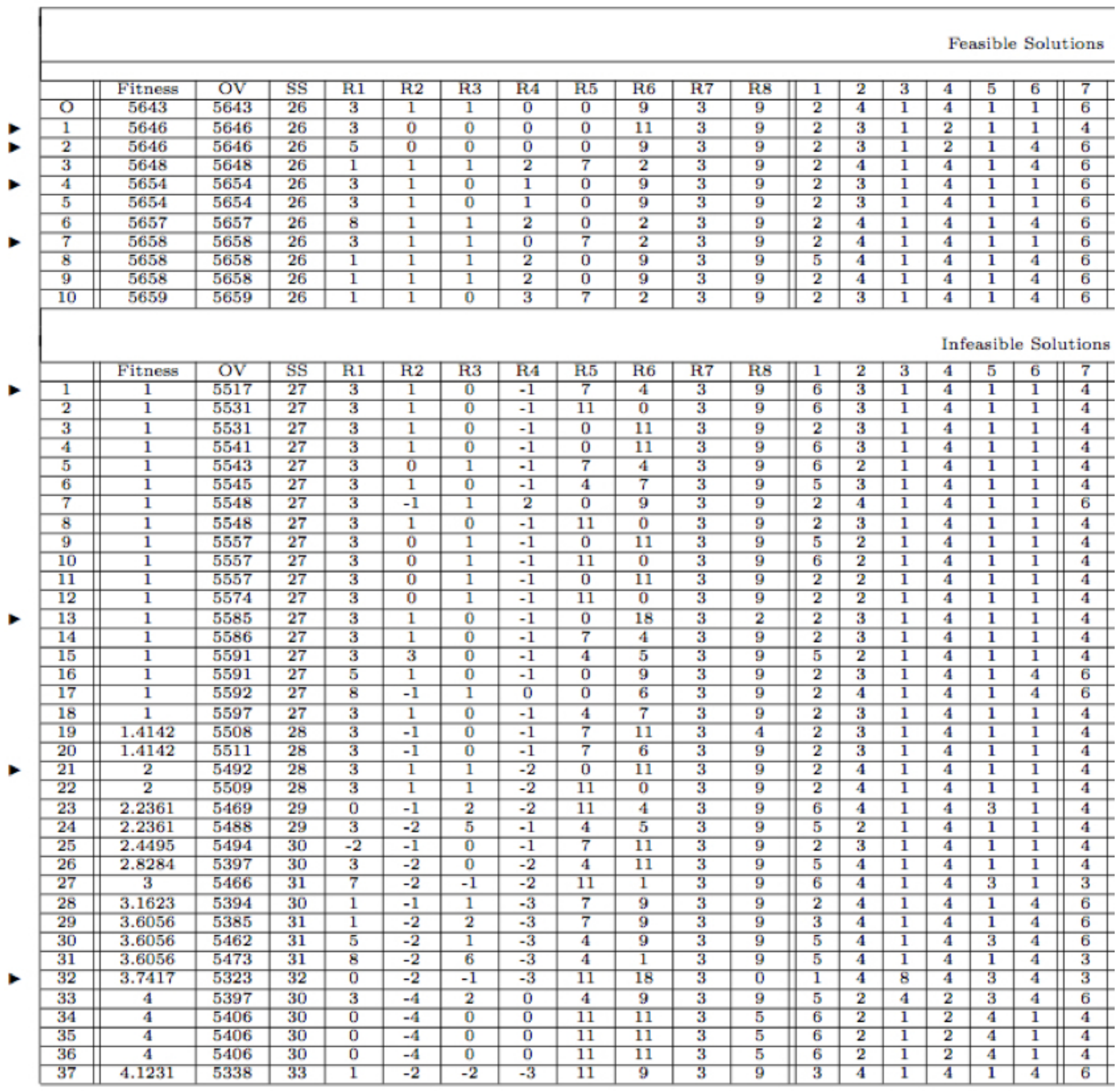
8	9	10	11	12	13	14	15	16	17	18	19	20
5	5	1	1	5	2	5	2	1	5	2	5	1
5	5	1	1	5	3	5	3	1	5	3	5	1
5	5	1	1	5	4	5	4	1	5	4	5	1
5	5	2	2	5	1	5	1	2	5	5	5	1
5	5	3	3	5	1	5	1	3	5	5	5	1
5	5	4	4	5	1	5	1	4	5	5	5	1
5	5	2	2	5	1	5	1	2	5	2	5	1
5	5	3	3	5	1	5	1	3	5	3	5	1
5	5	4	4	5	1	5	1	4	5	4	5	1
5	5	1	1	5	2	5	2	1	5	5	5	1
5	5	1	1	5	3	5	3	1	5	5	5	1
5	5	1	1	5	4	5	4	1	5	5	5	1
5	5	2	2	5	1	5	1	5	5	1	5	2
5	5	3	3	5	1	5	1	5	5	1	5	3
5	5	1	1	5	2	5	2	5	5	2	5	1
5	5	1	1	5	3	5	3	5	5	3	5	1
5	5	2	1	5	1	5	2	2	5	2	5	1
5	5	3	1	5	1	5	3	3	5	3	5	1
5	5	4	1	5	1	5	4	4	5	4	5	1

8	9	10	11	12	13	14	15	16	17	18	19	20
5	5	2	2	5	1	5	1	5	5	1	5	2
5	5	2	2	5	5	5	1	1	5	1	5	2
5	5	4	4	5	5	5	1	1	5	1	5	4
5	5	2	1	5	5	5	1	2	5	2	5	1
5	5	3	1	5	5	5	1	3	5	3	5	1
5	5	4	1	5	5	5	1	4	5	4	5	1
5	5	1	1	5	5	5	2	2	5	2	5	1
5	5	1	1	5	5	5	3	3	5	3	5	1
5	5	2	2	5	1	5	1	5	5	1	5	2
5	5	1	1	5	2	5	2	1	5	5	5	1
5	5	1	1	5	3	5	3	1	5	5	5	1
5	5	1	1	5	4	5	4	1	5	5	5	1
5	5	1	1	5	2	5	2	5	5	2	5	1
5	5	1	1	5	3	5	3	5	5	3	5	1
5	5	1	1	5	4	5	4	5	5	4	5	1
5	5	2	2	5	1	5	1	5	5	1	5	2
5	5	3	3	5	1	5	1	5	5	1	5	3
5	5	2	2	5	1	5	1	5	5	1	5	2
5	5	1	1	5	2	5	2	5	5	2	5	1
5	5	1	1	5	3	5	3	5	5	3	5	1
5	5	2	2	5	1	5	1	5	5	1	5	2
5	5	1	1	5	2	5	2	5	5	2	5	1
5	5	1	1	5	3	5	3	5	5	3	5	1
5	1	2	2	5	1	5	1	5	5	5	5	2
5	2	1	1	5	2	5	2	5	5	5	5	1
5	3	1	1	5	3	5	3	5	5	5	5	1
5	5	1	2	5	2	5	1	5	5	1	5	2
5	5	1	1	5	3	5	3	5	5	5	5	1
5	5	2	2	5	5	5	1	2	5	1	5	1
5	5	2	2	5	5	5	1	2	5	1	5	1
5	5	4	4	5	5	5	1	4	5	1	5	1

Figure 8: Elloumi Test Problem 2005De Solutions of Interest (Second Half Table)

1.9.5. A Medium Size Crossdock Assignment Problem

One of the practical areas where a DDSS can successfully be applied is Supply Chain Management, more specifically in the product distribution process. Traditionally warehouses, in carefully selected geographical hubs, are used to distribute products shipments from the production facilities to end users. In the late 1980s, Walmart and P&G came up with an



Feasible Solutions

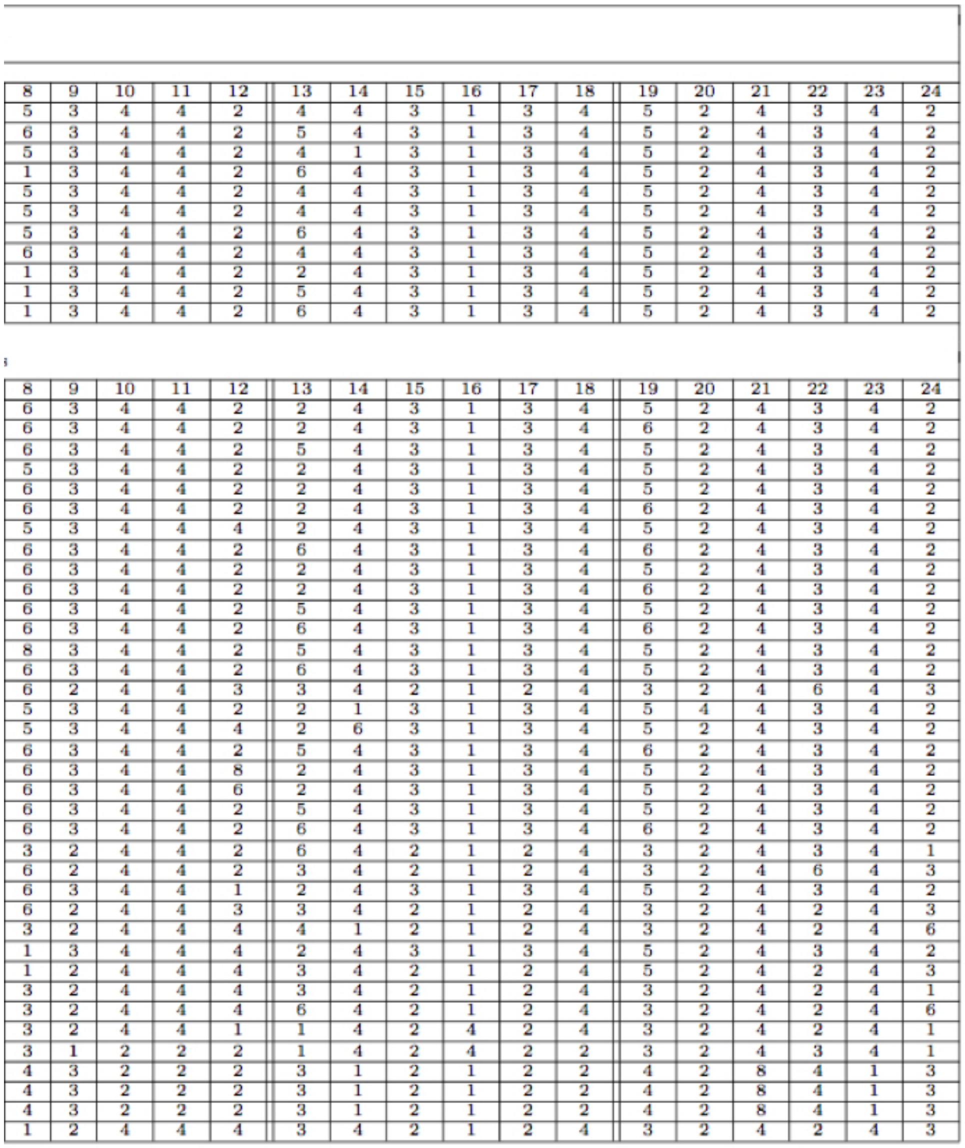
		Fitness	OV	SS	R1	R2	R3	R4	R5	R6	R7	R8	1	2	3	4	5	6	7
	O	5643	5643	26	3	1	1	0	0	9	3	9	2	4	1	4	1	1	6
▶	1	5646	5646	26	3	0	0	0	0	11	3	9	2	3	1	2	1	1	4
▶	2	5646	5646	26	5	0	0	0	0	9	3	9	2	3	1	2	1	4	6
	3	5648	5648	26	1	1	1	2	7	2	3	9	2	4	1	4	1	4	6
▶	4	5654	5654	26	3	1	0	1	0	9	3	9	2	3	1	4	1	1	6
	5	5654	5654	26	3	1	0	1	0	9	3	9	2	3	1	4	1	1	6
	6	5657	5657	26	8	1	1	2	0	2	3	9	2	4	1	4	1	4	6
▶	7	5658	5658	26	3	1	1	0	7	2	3	9	2	4	1	4	1	1	6
	8	5658	5658	26	1	1	1	2	0	9	3	9	5	4	1	4	1	4	6
	9	5658	5658	26	1	1	1	2	0	9	3	9	2	4	1	4	1	4	6
	10	5659	5659	26	1	1	0	3	7	2	3	9	2	3	1	4	1	4	6

Infeasible Solutions

		Fitness	OV	SS	R1	R2	R3	R4	R5	R6	R7	R8	1	2	3	4	5	6	7
▶	1	1	5517	27	3	1	0	-1	7	4	3	9	6	3	1	4	1	1	4
	2	1	5531	27	3	1	0	-1	11	0	3	9	6	3	1	4	1	1	4
	3	1	5531	27	3	1	0	-1	0	11	3	9	2	3	1	4	1	1	4
	4	1	5541	27	3	1	0	-1	0	11	3	9	6	3	1	4	1	1	4
	5	1	5543	27	3	0	1	-1	7	4	3	9	6	2	1	4	1	1	4
	6	1	5545	27	3	1	0	-1	4	7	3	9	5	3	1	4	1	1	4
	7	1	5548	27	3	-1	1	2	0	9	3	9	2	4	1	4	1	1	6
	8	1	5548	27	3	1	0	-1	11	0	3	9	2	3	1	4	1	1	4
	9	1	5557	27	3	0	1	-1	0	11	3	9	5	2	1	4	1	1	4
	10	1	5557	27	3	0	1	-1	11	0	3	9	6	2	1	4	1	1	4
	11	1	5557	27	3	0	1	-1	0	11	3	9	2	2	1	4	1	1	4
	12	1	5574	27	3	0	1	-1	11	0	3	9	2	2	1	4	1	1	4
▶	13	1	5585	27	3	1	0	-1	0	18	3	2	2	3	1	4	1	1	4
	14	1	5586	27	3	1	0	-1	7	4	3	9	2	3	1	4	1	1	4
	15	1	5591	27	3	3	0	-1	4	5	3	9	5	2	1	4	1	1	4
	16	1	5591	27	5	1	0	-1	0	9	3	9	2	3	1	4	1	4	6
	17	1	5592	27	8	-1	1	0	0	6	3	9	2	4	1	4	1	4	6
	18	1	5597	27	3	1	0	-1	4	7	3	9	2	3	1	4	1	1	4
	19	1.4142	5508	28	3	-1	0	-1	7	11	3	4	2	3	1	4	1	1	4
	20	1.4142	5511	28	3	-1	0	-1	7	6	3	9	2	3	1	4	1	1	4
▶	21	2	5492	28	3	1	1	-2	0	11	3	9	2	4	1	4	1	1	4
	22	2	5509	28	3	1	1	-2	11	0	3	9	2	4	1	4	1	1	4
	23	2.2361	5469	29	0	-1	2	-2	11	4	3	9	6	4	1	4	3	1	4
	24	2.2361	5488	29	3	-2	5	-1	4	5	3	9	5	2	1	4	1	1	4
	25	2.4495	5494	30	-2	-1	0	-1	7	11	3	9	2	3	1	4	1	1	4
	26	2.8284	5397	30	3	-2	0	-2	4	11	3	9	5	4	1	4	1	1	4
	27	3	5466	31	7	-2	-1	-2	11	1	3	9	6	4	1	4	3	1	3
	28	3.1623	5394	30	1	-1	1	-3	7	9	3	9	2	4	1	4	1	4	6
	29	3.6056	5385	31	1	-2	2	-3	7	9	3	9	3	4	1	4	1	4	6
	30	3.6056	5462	31	5	-2	1	-3	4	9	3	9	5	4	1	4	3	4	6
	31	3.6056	5473	31	8	-2	6	-3	4	1	3	9	5	4	1	4	1	4	3
▶	32	3.7417	5323	32	0	-2	-1	-3	11	18	3	0	1	4	8	4	3	4	3
	33	4	5397	30	3	-4	2	0	4	9	3	9	5	2	4	2	3	4	6
	34	4	5406	30	0	-4	0	0	11	11	3	5	6	2	1	2	4	1	4
	35	4	5406	30	0	-4	0	0	11	11	3	5	6	2	1	2	4	1	4
	36	4	5406	30	0	-4	0	0	11	11	3	5	6	2	1	2	4	1	4
	37	4.1231	5338	33	1	-2	-2	-3	11	9	3	9	3	4	1	4	1	4	6

Figure 9: Elloumi Test Problem 2408Aa Solutions of Interest (First Half Table)

innovative idea to adapt cross-docking operations (which were first pioneered in the US trucking industry in the 1930s) in order to streamline the supply chain from point of origin to point of sale. Cross-docking is a logistics technique and practice of unloading product shipments from incoming trucks or rail cars and loading them directly into outbound trucks or rail cars with little or no storage in between. Shipments typically spend less than twenty-four hours in a cross-dock, sometimes less than an hour (John J. Bartholdi and Gue [2004]). Retail cross-docking reduces operational costs by eliminating the need of high-maintenance



8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
5	3	4	4	2	4	4	3	1	3	4	5	2	4	3	4	2
6	3	4	4	2	5	4	3	1	3	4	5	2	4	3	4	2
5	3	4	4	2	4	1	3	1	3	4	5	2	4	3	4	2
1	3	4	4	2	6	4	3	1	3	4	5	2	4	3	4	2
5	3	4	4	2	4	4	3	1	3	4	5	2	4	3	4	2
5	3	4	4	2	4	4	3	1	3	4	5	2	4	3	4	2
5	3	4	4	2	6	4	3	1	3	4	5	2	4	3	4	2
6	3	4	4	2	4	4	3	1	3	4	5	2	4	3	4	2
1	3	4	4	2	2	4	3	1	3	4	5	2	4	3	4	2
1	3	4	4	2	5	4	3	1	3	4	5	2	4	3	4	2
1	3	4	4	2	6	4	3	1	3	4	5	2	4	3	4	2

8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
6	3	4	4	2	2	4	3	1	3	4	5	2	4	3	4	2
6	3	4	4	2	2	4	3	1	3	4	6	2	4	3	4	2
6	3	4	4	2	5	4	3	1	3	4	5	2	4	3	4	2
5	3	4	4	2	2	4	3	1	3	4	5	2	4	3	4	2
6	3	4	4	2	2	4	3	1	3	4	5	2	4	3	4	2
6	3	4	4	2	2	4	3	1	3	4	6	2	4	3	4	2
5	3	4	4	4	2	4	3	1	3	4	5	2	4	3	4	2
6	3	4	4	2	6	4	3	1	3	4	6	2	4	3	4	2
6	3	4	4	2	2	4	3	1	3	4	5	2	4	3	4	2
6	3	4	4	2	2	4	3	1	3	4	6	2	4	3	4	2
6	3	4	4	2	5	4	3	1	3	4	5	2	4	3	4	2
6	3	4	4	2	6	4	3	1	3	4	6	2	4	3	4	2
8	3	4	4	2	5	4	3	1	3	4	5	2	4	3	4	2
6	3	4	4	2	6	4	3	1	3	4	5	2	4	3	4	2
6	2	4	4	3	3	4	2	1	2	4	3	2	4	6	4	3
5	3	4	4	2	2	1	3	1	3	4	5	4	4	3	4	2
5	3	4	4	4	2	6	3	1	3	4	5	2	4	3	4	2
6	3	4	4	2	5	4	3	1	3	4	6	2	4	3	4	2
6	3	4	4	8	2	4	3	1	3	4	5	2	4	3	4	2
6	3	4	4	6	2	4	3	1	3	4	5	2	4	3	4	2
6	3	4	4	2	5	4	3	1	3	4	5	2	4	3	4	2
6	3	4	4	2	6	4	3	1	3	4	6	2	4	3	4	2
3	2	4	4	2	6	4	2	1	2	4	3	2	4	3	4	1
6	2	4	4	2	3	4	2	1	2	4	3	2	4	6	4	3
6	3	4	4	1	2	4	3	1	3	4	5	2	4	3	4	2
6	2	4	4	3	3	4	2	1	2	4	3	2	4	2	4	3
3	2	4	4	4	4	1	2	1	2	4	3	2	4	2	4	6
1	3	4	4	4	2	4	3	1	3	4	5	2	4	3	4	2
1	2	4	4	4	3	4	2	1	2	4	5	2	4	2	4	3
3	2	4	4	4	3	4	2	1	2	4	3	2	4	2	4	1
3	2	4	4	4	6	4	2	1	2	4	3	2	4	2	4	6
3	2	4	4	1	1	4	2	4	2	4	3	2	4	2	4	1
3	1	2	2	2	1	4	2	4	2	2	3	2	4	3	4	1
4	3	2	2	2	3	1	2	1	2	2	4	2	8	4	1	3
4	3	2	2	2	3	1	2	1	2	2	4	2	8	4	1	3
4	3	2	2	2	3	1	2	1	2	2	4	2	8	4	1	3
1	2	4	4	4	3	4	2	1	2	4	3	2	4	2	4	3

Figure 10: Elloumi Test Problem 2408Aa Solutions of Interest (Second Half Table)

cost warehouses and replacing them by product shipment exchange points, or Cross-Docking hubs.

This section demonstrates how the DDSS is applied to a real-life medium size cross-docking case. The following sections describe the given case, the application, and the findings in details.

- There are thirty trucks, T1 to T15 are inbound and T16 to T30 are outbound.
- There are twelve doors in the crossdock. Six on each side. D1 to D6 on one side for incoming truck and D7 to D12 on the other side for outgoing trucks.
- The flows from incoming trucks to outgoing trucks are specified in matrix F.
- The distance between each pair of inbound and outbound doors are known, D.
- For each truck, the volume of goods contained, either to be unloaded at an inbound door or loaded at an outbound door, is specified as vector V.
- For every door, the total volume of goods that can be unloaded or loaded during a shift is given as vector Q.
- Each truck must be assigned to one door.
- No installation fee for this crossdock.

The goal is to find the minimum-cost assignment of each truck to exactly one door, subject to the door capacity.

To define the problem, we use the following notation:

- M: a set of trucks. Let $I,\ J = \{1, ..., m\}$ index the set of trucks.
- N: a set of doors. Let $H,\ K = \{1, ..., n\}$ index the set of doors.
- x_{ik}: the decision variables x_{ik} are set to 1 if truck i is assigned to door k, 0 otherwise.
- v_i: the volume to be loaded/unloaded of truck i.
- q_k: the capacity of door k for handling loading/unloading during a shift.
- d_{kh}: the distance between door k and h, $d_{kh} = d_{hk}$.
- f_{ij}: the flow from incoming truck i to outgoing truck j.

The Crossdock problem is formulated as follows:

$$\min_{x_{ik}} \quad \sum_{i=1}^{\frac{m}{2}} \sum_{j=\frac{m}{2}+1}^{m} \sum_{k=1}^{\frac{n}{2}} \sum_{h=\frac{n}{2}+1}^{n} f_{ij} d_{kh} x_{ik} x_{jh} \tag{1.12}$$

$$subject\ to: \quad \sum_{k \in N} x_{ik} = 1 \qquad \forall\, i \in M, \tag{1.13}$$

$$\sum_{i \in M} v_i x_{ik} \le q_k \qquad \forall\, k \in N, \tag{1.14}$$

$$x_{ik} \in \{0,1\} \qquad \forall i \in N,\ \forall k \in M. \tag{1.15}$$

The constraints, including the integrality condition on the variables, state that each truck is assigned to exactly one door, and that the volume limit of each door is not exceeded. Matrices $\mathbf{F}$ and $\mathbf{D}$ and vector $\mathbf{V}$ and $\mathbf{Q}$ with elements f_{ij}, d_{kh}, v_i, and q_k are the parameters of the model. Matrix $\mathbf{X}$ with element x_{ik} is the decision variable.

Solution Representation, GA Operators for Crossdock Problem, and Experiment Setups

Our solution representation for the crossdock problem is similar to that of the CTAP except that one needs to distinguish the inbound/outbound trucks/doors since the crossdock problem has the constraint which allows only inbound trucks to be assigned to inbound doors (and only outbound trucks assigned to outbound doors). Given that there are a set of m trucks and a set of n doors, let $\mathbf{x} = x_1 x_2 \cdots x_{m-1} x_m$ be a solution for the crossdock problem. Let I = { 1, $\cdots$, m' } index the inbound trucks and I = { m'+1, $\cdots$, m} index the outbound trucks. Let K = { 1, $\cdots$, n' } index the inbound doors and K = { n'+1, $\cdots$, n} index the outbound doors. For our GA to comply with the inbound/outbound assignment requirements, we modify three of its operators:

- Random Initialization: for a solution string, the first m' entries are randomly assigned with numbers ranging between $[1..n']$ while the rest m - m' entries are randomly assigned with numbers ranging between $[n' + 1 \;..\; n]$.

- Mutation Operator: for a solution string $\mathbf{x}$, for any entry $x_i \in \mathbf{x}$ to mutate, the changed value ranges between $[1..n']$ if $i \leq m'$; or range between $[n' + 1 \;..\; n]$ if $i \geq m' + 1$.

- Crossover Operator: for a solution string $\mathbf{x}$, should a crossover to happen at the i^{th} position of the string, a local-path crossover will take place for only the substring consisted with $\mathbf{x}$'s first m' entries if $i \leq m'$; or the local-path crossover will take place for only the substring consisted with $\mathbf{x}$'s $m' + 1^{th}$ to m^{th} entries otherwise.

The constraint of assigning inbound (outbound) truck only to inbound (outbound) door is enforced with the above operator modifications. The solution representation and its correspondingly tailored GA operators reduces the size of solution space to $6^{30}(\approx 2.21 \times 10^{23})$.

In order to speed up the search process, we seed the initialization population with the top three feasible solutions found during our parameter tuning process. For the given medium size crossdock problem, the GA parameter setting is as follows: population size:300, generations:3000, trials:10, crossover:0.3, mutation:0.04, feasible/infeasible heap size:100, feasible threshold:0, infeasible threshold:20 (<0.5% of total volume limit).

Experiment Results

The DDSS solves this problem to the known optimal 13567. The found optimal solution utilizes only ten of the twelve doors. Table 17 shows that the optimal solution doesn't need inbound door #6 and outbound door #12. Seven other suboptimal solutions achieve less than 0.001% of total cost increase while offering different spared loading/unloading volume arrangement across different doors.

				Resource Usage											
				Inbound Door						Outbound Door					
SID	Fitness	Obj.Val.	subOpt%	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12
O	13567	13567	0	6	52	14	44	16	340	80	14	6	29	3	340
1	13568	13568	-0.0074	6	52	14	44	16	340	80	14	6	18	14	340
2	13571	13571	-0.0295	93	44	14	56	15	250	180	29	2	3	3	255
3	13571	13571	-0.0295	250	15	56	14	44	93	255	3	3	2	29	180
4	13574	13574	-0.0516	37	21	14	44	16	340	80	14	6	29	3	340
5	13575	13575	-0.0590	37	21	14	44	16	340	80	14	6	18	14	340
6	13580	13580	-0.0958	93	44	14	56	96	169	180	29	2	3	3	255

Table 16: GQAP: Near-Optimal Feasible Solutions Information

	Inbound Trucks														
SID	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15
O	3	1	1	5	4	2	1	5	2	2	4	3	1	5	5
1	3	1	1	5	4	2	1	5	2	2	4	3	1	5	5
2	3	6	5	1	2	4	5	1	4	5	2	3	4	5	1
3	4	1	2	6	5	3	2	6	3	2	5	4	3	2	6
4	3	1	1	5	4	2	2	5	2	1	4	3	1	5	5
5	3	1	1	5	4	2	2	5	2	1	4	3	1	5	5
6	3	6	6	1	2	4	5	1	4	5	2	3	4	5	1
	Outbound Trucks														
SID	T16	T17	T18	T19	T20	T21	T22	T23	T24	T25	T26	T27	T28	T29	T30
O	9	10	10	9	8	7	11	7	7	9	9	10	11	11	8
1	9	11	11	9	8	7	11	7	7	9	9	10	10	11	8
2	9	7	8	9	11	12	7	10	11	8	10	9	8	11	10
3	10	12	11	10	8	7	12	9	8	11	9	10	11	8	9
4	9	10	10	9	8	7	11	7	7	9	9	10	11	11	8
5	9	11	11	9	8	7	11	7	7	9	9	10	10	11	8
6	9	7	8	9	11	12	7	10	11	8	10	9	8	11	10

Table 17: GQAP: Near/Optimal Feasible Solutions Truck Assignment

Seven infeasible solutions of interest are listed in Table 19. One can reduce the total cost by 8 points if additional 12 units of unloading volume (a 3.5% volume increase of door #2, a 0.3% of total volume of this crossdock) is available for inbound door #2. It is also achievable to further reduce the total cost by 26 points if 7 and 12 extra unloading volumes can be handled for inbound door # 1 and #2 respectively (solution#3) . Solution #7 achieves 29 points deduction on the total cost by increasing 25 loading volume at door #2 (a 0.61% increase on total volume).

				Resource Usage											
				Inbound Door						Outbound Door					
SID	Fitness	Obj.Val.	OV Imp%	D1	D2	D3	D4	D5	D6	D7	D8	D9	D10	D11	D12
1	12	13559	0.0590	70	-12	14	44	16	340	80	14	6	29	3	340
2	12	13560	0.0516	70	-12	14	44	16	340	80	14	6	18	14	340
3	13.892	13541	0.1916	-7	-12	14	44	93	340	18	14	6	29	65	340
4	13.892	13542	0.1843	-7	-12	14	44	93	340	18	14	6	18	76	340
5	13.892	13555	0.0885	-7	-12	14	44	107	326	18	14	6	29	65	340
6	13.892	13556	0.0811	-7	-12	14	30	107	340	18	14	6	18	76	340
7	25	13538	0.2138	6	-25	14	44	93	340	18	14	6	29	65	340

Table 18: GQAP: Near-Optimal Infeasible Solutions Information

	Inbound Trucks														
SID	T1	T2	T3	T4	T5	T6	T7	T8	T9	T10	T11	T12	T13	T14	T15
1	3	1	1	5	4	2	1	5	2	2	4	3	2	5	5
2	3	1	1	5	4	2	1	5	2	2	4	3	2	5	5
3	3	1	1	5	4	2	1	5	2	2	4	3	2	1	5
4	3	1	1	5	4	2	1	5	2	2	4	3	2	1	5
5	3	1	1	6	4	2	1	5	2	2	4	3	2	1	5
6	3	1	1	4	4	2	1	5	2	2	4	3	2	1	5
7	3	1	1	5	4	2	1	5	2	2	4	3	1	2	5

	Outbound Trucks														
SID	T16	T17	T18	T19	T20	T21	T22	T23	T24	T25	T26	T27	T28	T29	T30
1	9	10	10	9	8	7	11	7	7	9	9	10	11	11	8
2	9	11	11	9	8	7	11	7	7	9	9	10	10	11	8
3	9	10	10	9	8	7	11	7	7	9	9	10	11	7	8
4	9	11	11	9	8	7	11	7	7	9	9	10	10	7	8
5	9	10	10	9	8	7	11	7	7	9	9	10	11	7	8
6	9	11	11	9	8	7	11	7	7	9	9	10	10	7	8
7	9	10	10	9	8	7	11	7	7	9	9	10	11	7	8

Table 19: GQAP: Near Optimal Infeasible Solutions Truck Assignment

1.10. Summary and Discussion

To summarize this chapter:

- With regard to the post-solution deliberation / analysis, we introduce the Solutions of Interest (SoI) problem, i.e., that of finding non-optimal solutions of interest for solving constrained optimization problems. We further identify the two subproblems, the problem of finding feasible solutions of interest (FoIs) and the problem of finding infeasible solutions of interest (IoIs), and illustrate the importance of the SoI problem for the study of constrained optimization by metaheuristics, including particularly evolutionary computations.
- For finding the SoIs, we choose genetic algorithms (GA) as our searching method due to their population-based ameliorating characteristic. In the preliminary study, we compare two common GA approaches: the penalty GA and the multi-population GA. We find that Two Population GA (FI-2Pop GA) performs better on sampling various SoIs than the Penalty GA does. (Our experience with penalty GAs suggests that they are not very good at finding IoIs, which is hardly surprising.)
- We propose the prioritized solution collecting procedure and identify four types of non-optimal solutions of interest (SoIs):
  - For the feasible solutions of interest (FoIs), we have the following two solution types:
    - Near-Optimal Solutions - the `FoI(Obj)`s: feasible solutions with objective values close to the known optimal value.
    - High-slack Solutions - the `FoI(Slacks|MinObj)`s: feasible solutions with high amount of slack sum on the constrained resources, given that their objective values are no worse than the specified threshold.
  - For the infeasible solutions of interest (IoIs), we have the following two solution

types:

  * Near-Feasible Solution - the `IoI(SumV)`s: infeasible solutions that are very close to the feasibility boundary.

  * High-Objective-Value Solution - the `IoI(Obj| MaxDist)`s: infeasible solutions that promise high objective values, provided that they are not too far away (within a specified distance) from the feasibility boundary.

- We demonstrate the usefulness of the SoIs (FoIs and IoIs) for post-solution deliberation with a prototypical problem, the Beasley OR-Library GAP4-2. Based on extensive test runs on different problems, we believe that these reported results are representative.

- We demonstrate the effectiveness of the proposed approach on finding SoIs with not only GAPs but also GQAPs.

- We observe that the two population GA display a gradual, noisy leveling off in finding new SoIs throughout the multiple trials. As expected, the required computational effort for getting the SoIs is typically larger than if we simply look for the optimal solution. Our approach can serve as a benchmark for other methods.

Clearly, very much additional research is called for on the SoI problem. Our aim in this chapter is to motivate and frame such prospective research. For future research, besides exploring other kinds of models and gaining practical experience, we see a number of especially important agenda items:

1. Based on our experience, the FI-2Pop GA is a credible approach to finding SoIs. Other approaches that work as well or better must be sought and investigated.

2. There is no reason to limit the SoIs searching process with a single method. Multiple heuristics might be employed. Hyperheuristics (Burke et al. [2003]), for example, offer a framework for selecting, combining, generating, or adapting multiple heuristics (or components of such heuristics) to obtain the Sols.

3. Searches that focus on regions of interest may provide the decision maker with a more complete inventory of available options. Interpolation, neighborhood search, simulated annealing, etc., among the initial SoIs (found by whatever methods) may well be effective to accomplish such a task.

4. Other SoIs collecting procedures should be explored. Our current proposal categorizes the SoIs into four types and searches different type of solutions separately. Perhaps different SoI categorization or some kind of combined, weighted search would be preferable, i.e., to integrate the degree of infeasibility into the objective function (which can be implemented with Guided Local Search (Voudouris and Tsang [2003]), in which different features of infeasibility are treated as penalty terms that augment the objective function).

5. In our searches, we used two heaps each for the feasibles and the infeasibles, one heap for each "objective" (fitness value, slack value). Perhaps a combined, weighted search would suffice or be preferable, i.e., to construct a heap that integrates the degree of infeasibility into the objective function. One way to implement this search is via Guided Local Search (Voudouris and Tsang [2003]), in which different features of infeasibility are treated as penalty terms that augment the objective function.

CHAPTER 2 : Considering Additional Objectives with the Metaheuristic Approach

2.1. Introduction

In this chapter, we propose and explore the heuristic approach for multi-objective optimization, in the context of two variants of two-sided matching problems—Stable Marriage Problems and Couples Problems.

Widely encountered in practice, the two-sided matching problem involves two disjoint sets of agents who have preferences over possible matchings between members of opposite sets; and the objective is to find a 'good' match. The Stable Marriage Problem is prototypical of two-sided matching problems. In the standard form, variants of the well-known Gale-Shapley deferred acceptance algorithm (GS/DAA) are used to find stable matches. Using an evolutionary heuristic, the Stable Marriage Problem is investigated as a multi-objective problem, considering social welfare, equity (or fairness), and regret in addition to stability as important aspects of any proposed match. This study finds that the evolutionary heuristic is a reliable approach to discover matches that are Pareto superior to those found by the GS/DAA procedure. Ramifications of this finding are briefly explored, including the question of whether stability in a matching is often strictly required.

Structure of the chapter. Section 2.2 provides an overview of the stable marriage matching problem, its variants, and the widely known algorithms which have been used for many years in practice for finding a stable matching. Section 2.3 presents the proposed evolutionary computation approach and study its performance with the stable marriage problem. Section 2.4 describes a generalized version of Stable Marriage Problems, the Couples Problems, and examines the applicability of the proposed approach in settings for the more generalized matching problem. Finally, Section 2.5 contains concluding remarks.

2.2. Motivation and Related Work

In the usual case, markets are distributed, with buyers and sellers mostly on their own to find each other and to negotiate terms of trade, however, distributed markets may fail in one way or another. A common response is to create a centralized market organized by a third party whose responsibility is to set the conditions of trade, for example the price, based on the bids and asks from the buyers and sellers. Many electricity markets are organized in this way. For example, the deregulated electricity producers offer supply schedules to a third party, often called the independent systems operator or ISO, who aggregates the supply schedules, observes the market demand, and sets the price of electricity (for a given period of time). Another common example is clearing houses between international telecommunication operators for roaming tariffs and settlements.

The market concept is more general than just settling financial transactions or exchanging goods between buyer and seller. The admission processes for Boston public schools and New York City high schools and the hospital internship programs are also examples of centralized clearinghouse. No matter whether it is about the electricity/telecommunication markets, school admissions, or physician appointments, these cases are all examples of the two-sided matching problem, the problem which is the subject of this chapter.

In a two-sided matching problem (Roth and Sotomayor [1990]), we are given two sets (sides, X and Y) of individuals and asked to form pairs consisting of one member from each set. Common examples dealt with in practice include pairing students with schools, men with women, workers with employers and so on. Different matching problems come with different requirements on the agent sets (X and Y) and the matching μ (or M). For example, for a simple marriage problem, the number of members in both sets are required to be the same and each member of X must be paired with exactly one member of Y. On the other hand, for a college admissions problem, one set of the agents, X (the student), is much larger than the other, Y (the school), and each member of X can be paired with at most one member of Y (a student attends only one school) while a member of Y can be paired with multiple members of X (a school accepts a certain quota of students). Roth [2008] notes that the two-sided matching models are often natural for representing markets, in which agents need to be paired up. As many of the decentralized or free markets experience failure and unsatisfactory performance such as unraveling (e.g., offers to students are made earlier and earlier), congestion (e.g., offerers don't have enough time to make new offers after prior offers are rejected), and participants' disruptive strategic behavior, a growing number of those markets are replaced by the centralized ones, in which a coordinating agency undertakes periodic matching between two sides of a specific market. Some examples include NRMP (National Resident Matching Program, NRMP) of US, CARMS (Canadian Resident Matching Scheme, CaRMS) of Canada, SFAS (Scottish Foundation Allocation Scheme[1],Irving) of Scotland, and JRMP (Japan Resident Matching Program, JRMP) of Japan (see Roth [2008], Iwama and Miyazaki [2008]). In these markets, typically, the individual doctors submit their rankings of hospitals, the hospitals submit their rankings of doctors, and a third party organization undertakes matching doctors with hospitals.

A presumption in matching problems (as distinguished from assignment problems, which are treated in operations research and employ non-strategic decision making) is that the participating agents on both sides have preferences and interests of their own and all agents have the capacities to act based on their own interests. Consequently, matches are typically evaluated in terms of stability; a match is said to be stable if there is no unmatched pair of individuals who would prefer to be matched to each other than to stay with their current partners. In other words, in a stable match, for any specific x which is not paired with a specific y, either such x prefers its own partner to such y or such y prefers its own partner to such x. Requiring matches to be stable in the first place will prevent reformation among pairs and its attendant costs.

While stability is desired, there are other aspects of a matching which should also be emphasized. Kojima [2012] establishes the impossibilities for affirmative action under any stable

[1]Before 2006, the allocation was to Pre-Registration House Officer posts, or PRHOs, the scheme was known as SPA – Scottish PRHO Allocations.

matching mechanism, such as those used in the admission processes for Boston and New York City high schools. Nakamura et al. [1995] use a genetic algorithm to find gender-unbiased solutions for the stable marriage problem. Two-sided matching problems, and the stable marriage problem in particular, have received exploratory investigation as dual objective problems in Fuku et al. [2006], Vien and Chung [2006], Axtell and Kimbrough [2008]. Dasgupta et al. [2008], Deb [2001], Coello Coello et al. [2007] offer useful findings pertaining to multi-objective evolutionary algorithms generally. Aldershof and Carducci [1999] report optimistically on application of a genetic algorithm to two-sided matching problems, but the problems they examine are smaller than the 20×20 problems discussed here, so they do not address scaling issues. Vien et al. [2007] propose an innovative use of ant colony optimization for the stable marriage problem. These prior studies lead us to believe that population-based metaheuristics are very promising for two-sided matching problems seen as multi-objective problems.

The point of departure for this chapter is the observation that two-sided matches can be evaluated, and for many applications should be evaluated, according to multiple objectives, particularly stability, equity, social welfare, etc. Given that one would consider designing or even centralizing a matching market (as is widely done in practice), the question arises of how best to provide the market operators and users with match options that map the Pareto frontier (as well as possible) in at least these three objectives. In what follows we explore an evolutionary computation approach to this goal. We demonstrate our approaches with two different two-sided matching problems, the stable marriage problem and the couples problem. The results are then compared with what can be produced by the standard approach, the deferred acceptance algorithm of Gale and Shapley [1962]. Before discussing the experiment results, some background information is first presented.

2.2.1. Matching – Simple (Stable) Marriages Problem

In a two-sided matching problem, as discussed earlier but here with more detail, we are given two finite and disjoint sets (sides) of agents, X and Y, each of whom has complete and transitive preferences over the individuals in the other set. The objective is to find a matching, μ, consisting of a decision, "in or out?", for each pair (x, y), $x \in X$, $y \in Y$. It is helpful to view a matching, μ, as represented by a matrix, M, of size $|X| \times |Y|$, based upon arbitrary orderings of X and Y . The element m_{ij} of M equals 1, if $x_i \in X$ is matched with $y_j \in Y$; otherwise the element is 0. Thus the element m_{ij} of M represents the pair (x_i, y_j). Matchings pair up agents from X and Y. The following is the requirements of the simple marriage problem, a prototypical two-sided matching problem (Gusfield and Irving [1989], McVitie and Wilson [1971a,b], Iwama and Miyazaki [2008]).

About the agents: The two finite and disjoint sets of agents are $X = \{x_1, x_2, \cdots, x_n\}$ of women, and $Y = \{y_1, y_2, \cdots, y_n\}$ of men. The number of agents in both sets is required to be the same, i.e. $|X| = |Y| = n$.

About the preferences: Each woman has preferences over the men, and each man has preferences over the women. The preferences of each woman (man) is presented by an ordered list, $p(x)$ $(p(y))$, on the set Y (X). That is, a woman x's (man y's)preferences could be of the form $p(x) = y1, y2, \cdots, y_n$ $(p(y) = x_1, x_2, \cdots, x_n)$, indicating that her (his) first choice is to be married to man y_1 (woman x_1), her (his) second choice is to be married to man y_2 (woman x_2), etc. We write $x \succeq_y x'$ to mean y prefers x to x', and $y \succeq_x y'$ to mean x prefers y to y'. The preferences are transitive, strict, and complete. By saying transitive, we mean that if $y_1 \succeq_x y_2$ and $y_2 \succeq_x y_3$, then $y_1 \succeq_x y_3$. By saying *strict*, we mean that an agent is not indifferent between any two acceptable alternatives. By saying *complete*, we mean that every man $y \in Y$ is an acceptable alternative for every woman $x \in X$, and vice versa.

About the matching: Every participating agent prefers marrying someone from the opposite gender set to remaining single. Therefore, for any valid matching, each woman (or member of X) must be paired (or matched) with exactly one man (or member of Y), and vice versa. In terms of the matching matrix M, this means that there is one '1' in each row and one '1' in each column. M is thus an $n \times n$ permutation matrix and the number of possible valid matchings is $n!$. Let P be the set of preference lists, i.e.,

$P = \{p(x_1), \cdots, p(x_n), p(y_1), \cdots, p(y_n)\}$ (or $P_{i,i \in X \cup Y}$), one list for each participating agent.

We can now refer to a simple marriage market $\mathscr{M}$ as a tuple $(X, Y, P_{i,i \in X \cup Y})$.

2.2.2. Deferred Acceptance Algorithm

The deferred acceptance algorithm (DAA) was first published in a paper by Gale and Shapley [1962], although the procedure was discovered and used independently before. As described by Roth [2008], 'the algorithm works by having agents on one side of the market make proposals (offers or applications) to agents on the other, in order of preference. Those who receive more proposals than they can accept reject their least preferred, but do not immediately accept those they do not reject; instead, they hold them without commitment, and acceptances are deferred until the end of the algorithm. In the meantime, agents who have been rejected make new proposals, which lead to new rejections (including proposals that were held at an earlier period but are less preferred than a new proposal), until there are no rejected agents who wish to make further proposals. At this point all proposals that are being held are finally accepted, to produce a matching'. The pseudocode of DAA is presented in Figure 11.

DAA has some key properties worth highlighting. First, as proved by Gale and Shapley, under the special assumptions they made (e.g., preference ranking by agents, etc., which for the purpose of discussion we retain), the stable marriage problem and the admissions problem (see above) have stable matches and the DAA will find one and will find one quickly ($O(n^2)$). Second, the DAA is asymmetric. One side proposes, the other disposes. Focusing now on the marriage problem, if the men propose, they obtain a stable match that is male optimal in the sense that no man in this match strictly prefers (does better in) any other stable match. Conversely, the match is female pessimal in the sense that no woman is worse off in any other stable match. Vise versa, if the women propose, it produces the stable matching which is the best for women and the worst for men (Gale and Shapley [1962], Gusfield and Irving [1989], Knuth [1997]).

```
1. Assume: $|X| = |Y| = n$.
2. Each $x \in X$ ranks each $y \in Y$, and each $y \in Y$ ranks each $x \in X$.
3. Matched $\longleftarrow \emptyset$, Unmatched $\longleftarrow \emptyset$.
4. For each $y$, $string.y \longleftarrow [\,]$.
5. Each $x \in X$ proposes to its most-preferred $y$, appending $x$ to $string.y$.
6. Each $y$ with $length(string.y) > 1$ (i.e., with more than one proposal), retains in the
   string its most preferred member of the string, and removes the rest, adding them to
   $Unmatched$.
7. Do while $Unmatched \neq 0$:
   (a) Each $x \in Unmatched$ proposes to its most-preferred $y$, among the $Y$s that have
       not already rejected $x$, appending $x$ to $string.y$.
   (b) $Unmatched \longleftarrow \emptyset$.
   (c) Each $y$ with $length(string.y) > 1$ (i.e., with more than one proposeal), retains
       in the string its own most preferred member of the string, and removes the rest,
       adding them to $Unmatched$.
8. For each $x$ remaining on some $string.y$, $(x, y)$ is added to $Matched$.
9. Stop. Each $x$ is matched to a distinct $y$, who has $x$ as the sole member of its string.
   This is recorded in $Matched$.
```

Figure 11: Pseudocode for the deferred acceptance algorithm (DAA) for the simple marriage matching problem, Xs proposing to Ys.

Although here we consider it only in the context of the marriage problem, this asymmetry is a general characteristic of the DAA in its various forms. It raises the important question of whether better matches exist and can be found. To this end, we want to look at stable matches that may be preferable to the matches found by the DAA. As stated above, we want to examine both social welfare and equity. Further, it is natural to raise the question of multiple objectives in the context of nearly-stable matches, by which we mean matches with relatively few unstable pairs. Decision makers, including agents participating in a centralized market, may quite reasonably want to exchange some stability for improvements in, say, social welfare or equity. We note that in many cases it may be practically difficult, or made practically difficult by the operator of the centralized market, for members of matched couples to undertake swaps, regardless of their preferences.

For any given problem, these issues could be neatly resolved by finding all of the stable solutions and comparing them with respect to equity, social welfare, and whatever other measures of performance are relevant. Predictably, however, this is an intractable problem. Irving and Leather [1986] have shown that the maximum number of stable matches for the simple marriage matching problem grows exponentially in n (see also Gusfield and Irving [1989], Halldorsson et al. [2007]). Further, they provide a lower bound on the maximum by problem size. Remarkably, the lower bound is 104,310,534,400 for a problem as small as n = 32 (Irving and Leather [1986]). Further, they establish that the problem of determining the number of stable matches is #P-complete. These are, of course, extreme-case results. But very little is known about average cases. So we are left to rely upon heuristics, and we shall for the remainder of this chapter.

2.2.3. Matching With Couples – Hospital Residents Problem with Couples

There are other variations of two-sided matching problems common in everyday life (see Irving [1994], Iwama et al. [1999], Fleiner et al. [2007], Cechlarova and Fleiner [2005], Irving and Manlove [2002], Cechlarova and Ferkova [2004], Irving and Scot [2007], Echenique and Yenmez [2007]). A large and growing number of centralized markets apply the two-sided matching model, e.g., Roth [2008] lists over 40 labor markets, mostly in medical fields. Let's take the US National Resident Matching program (NRMP) as an example. As the twentieth-century American labor market transformed, there was increased labor force participation of married women leading to a growing number of two-career households. When a couple looked for two jobs, they were faced the coordination problem not only with each other but also with their prospective employers. In the 1950s, in response to persistent failures of decentralized means to organize the market in a timely and orderly way, the NRMP was established as a centralized labor market for medical school graduates in the US seeking their first employment (Roth [1990]). The original NRMP worked well for individual students but not for couples. Couples who desired residency positions geographically near to each other were forced to not use the centralized service and be matched under a special couples algorithm which may not have reflected the couple's true preferences. As the participation rates declined, NRMP modified its algorithm in 1983 to allow couples to submit preferences over pairs of residency positions (hence the Hospitals Residents Problem with couples is also known as 'the Couples Problem'). Unfortunately, many of the appealing mathematical results for the stable matching problem are not extended to the couples problem. Aldershof and Carducci [1996] summarizes the differences between the marriage problem and the couples problem as follows:

- Gale and Shapley's algorithm shows that there is always a stable matching for any preference list of the marriage problem. But there are instances of the couples problem that do not have a stable matching (Roth [1984]).

- For marriage problem, there is always a stable matching M that is hospital (student) optimal in the sense that every hospital (student) is at least as well off under M as it is under any other stable matching (Gale and Shapley [1962]). On the other hand, even if an instance of the couples problem has a stable matching, it may not have a hospital optimal or student optimal stable matching.

- Consider an instance of the couples problem in which the preference lists are not complete. There may be stable matchings which leave different numbers of positions unfilled.

Aldershof and Carducci illustrate the second and third points with a couples problem instance in which there are two couples and four residency positions. It shows that the improvement of usefulness of the NRMP to couples comes at the cost of some uncertainty in the results; particularly the possibility of different stable matchings yielding different sets of positions filled and of students assigned. Klaus et al. [2007] further report two problems that raise concern for the modified NRMP algorithm (which accommodates the couples problem):

- the new NRMP algorithm may not find an existing stable matching.

- the new NRMP algorithm is not strategy-proof, it may be manipulated by couples acting as singles.

After the crisis in confidence in 1990s, related organizations advocated reconsideration of the algorithm (also see Williams [1995], Peranson and Randlett [1995]) and the Board of Directors of NRMP commissioned the design of a new algorithm. The Roth-Peranson algorithm was introduced in 1998 to accommodate couples and other complications. The Roth-Peranson algorithm conceptual design is detailed in Roth and Peranson [1999]. When going beyond the simple matching problem, the optimal stable matches do not exist in general. The most concerned issues are related to the order in which applicants and programs are processed. Experiments are conducted and it is shown that:

- The sequencing differences do not cause substantial or predictable changes in the match result (e.g., applicants or programs selected first do not necessarily do better than their counterparts selected later).
- The sequence of processing does affect the likelihood that an algorithm will produce a stable matching (e.g., introduce couple last reduces the potential difficulty to find a stable matching without changing the prospects of couples or single applicants in the match).

In a later section, we will show how heuristics can help on searching for stable matches with an example couples problem.

2.2.4. Optimization Criteria for Matching

As mentioned previously, we note that in many cases it may be practically difficult, or made practically difficult by the operator of the centralized market, for members of matched couples to undertake swaps, regardless of their preferences. The difficulty of taking on a swap leads us to question the heightened importance of solution stability in the standard two-sided matching problem. What are the other objectives the decision makers should care about? In the case of 'nearly-stable' matching, by which we mean matches with relatively few unstable pairs, what are the other qualities of a solution that the decision maker may be willing to exchange for at the cost of some stability? Here, on top of solution stability, we will consider three other commonly addressed criteria (Iwama and Miyazaki [2008]): the fairness between the two matching sides, and the welfare and regret of the all participating agents .

Given that our two finite and disjoint sets of agents are X and Y - with number of agents n_X and n_Y respectively, and a matching $M = (X, Y)$. Let $x_i \in X$, $y_j \in Y$, and $r_{x_i}(y_j)$ be the rank of y_j on x_i's preference list. Since the most preferred candidate ranks 1, the lower rank number indicates the higher preference. Let's keep the assumption that $n_X = n_Y$ for now (otherwise, the welfare and fairness definitions won't be valid if $n_X \neq n_Y$).

- **Stability**: We define the stability of a solution as the number of unstable (blocking) pairs in a solution M. Let (x_1, y_1), (x_2, y_2) be a pair of matched agents. The pair is said to be unstable if x_1 prefers being matched with y_2 instead of y_1 meanwhile y_2 prefers being matched with x_1 than with x_2. i.e. $r_{x_1}(y_1) \succ r_{x_1}(y_2)$ *and* $r_{y_2}(x_2) \succ r_{y_2}(x_1)$. The same argument holds if x_2 prefers y_1 over y_2 while y_1 prefers x_2 over x_1; i.e. $r_{x_2}(y_2) \succ r_{x_2}(y_1)$ *and* $r_{y_1}(x_1) \succ r_{y_1}(x_2)$.

- **Welfare**: We measure the welfare of a solution by the average rank of that each agent is matched with, via summing the respective preference ranks of the matched agents; i.e.,
$$Welfare = \frac{\sum_{x_i \in (X,Y)} r_{x_i}(y_j) + \sum_{y_j \in (X,Y)} r_{y_j}(x_i)}{n_X + n_Y}$$
The lower welfare value indicates the better solution.

- **Fairness**: Fairness can be measured as the inequality in welfare distribution; i.e.,
$$Fairness = |*| \frac{\sum_{x_i \in (X,Y)} r_{x_i}(y_j)}{n_X} - \frac{\sum_{y_j \in (X,Y)} r_{y_j} r(x_i)}{n_Y}$$

High fairness score reflects a high inequality between the two matched sides. The lower fairness value indicates a more equal welfare distribution. A fairness score closer to 0 indicates a better solution.

- **Regret**: Regret is commonly measured as the largest rank value among all matched agents; i.e.,

  $$Regret = \max_{(x_i,y_j)\in(X,Y)} \max\{r_{x_i}(y_j), r_{y_j}(x_i)\}$$

  We define the regret score as the average value of the larger rank of each matched pair; i.e.,

  $$Regret = \frac{\sum_{x_i,y_j\in(X,Y)} \max\{r_{x_i}(y_j), r_{y_j}(x_i)\}}{|M|}$$

  Higher regret score reflects a higher dissatisfaction among the less happy agents in the matched pairs. The lower regret score indicates less grumbling agents in a better solution.

2.3. Evolutionary Matching for Stable Marriages Problem

We developed a genetic algorithm based system, called Stable Matching GA, that samples the solution space of the simple marriage matching problem. Solutions are presented as permutations of $1\cdots\ n$. After parameter tuning for best performance, we choose a mutation rate of 0.4, a two-point crossover rate of 0.6, a population size of 50, runs of 2,000 generations, and 100 repetitions (or trials) per problem. Mutation was effected by randomly choosing alleles at two loci and swapping them, thereby maintaining a valid solution, as a permutation. We used order crossover, OX, as our two point crossover operator (Michalewicz [1996] page 217, Goldberg [1989] page 174). The fitness of a given solution is its number of unstable pairs. The lower the fitness value, the better a solution is and the higher chance it has for participating in the reproduction process. For a given population, one generation of solutions goes through three processes: fitness evaluation, selection (we used tournament-2), and reproduction (with mutation and crossover as described above). The evolution stops after a specified number of generations are reached. We ran the GA seeking to minimize the number of unstable pairs in the match solutions.

2.3.1. Stable Matches

Here we report with regard to the stable matches found by the GA for these 25 random 20×20 problems, and we compare the GA's solutions with the DAA solutions for these problems. The results are summarized in Table 20. Here, in Table 20, the column labeled Case is for the 25 random 20×20 random problem instances; "D(DorE) 1G.S. Soln" means the count of strictly dominant (better than the DAA (Gale-Shapley) solutions on the two dimensions of fairness and social welfare) or (in parentheses) the count of weakly dominant (at least as good as the DAA solution)– for at least one of the solutions (and there may be only one). "D(DorE) 2G.S.Sol" means the count of strictly dominant (better than the DAA on the two dimensions of fairness and social welfare) or (in parentheses) the count of weakly dominant (at least as good as the DAA solutions) –f or both DAA solutions (although there may be only one). So the form $X(Y)/Z$ means X solutions strictly dominating, Y solutions weakly dominating, and Z solutions found by the GA overall. (All of these solutions are stable solutions.) Finally, Found G.S.Soln/#GS with the form X/Y means of the Y DAA solutions (from Gale-Shapley), X of them were found by the GA.

To summarize, Table 20 shows:

- In 18 of 24 cases, the GA found at least one stable solution that strictly dominates both of the GS/DAA solutions. (In case 20, there is only one GS/DAA solution.)
- Excluding case 20 with only one GS/DAA solution, in 23 of 23 cases the GA found one or more stable solution that strictly dominates one of the two GS/DSS solutions.

Our study thus shows promising results on using a GA to search for favorable matching schemes. In terms of searching for stable match schemes, the GA found either strictly better or at least equally good solutions compared to the Deferred Acceptance algorithm results with regard to fairness and social welfare. When stability is not the only objective, GAs provide many other dominant solutions. Depending on how different objectives are weighted, sometimes a minimum number of unstable pairs may be a cheap price to pay for the improvement on other objectives (for example, higher individual satisfaction or greater assignment fairness). The key point of our finding is about the GA's capability of providing decision makers with information about the otherwise unseen alternatives.

Test Case	D(DorE) 1G.S.Soln / TC	D(DorE) 2G.S.Soln / TC	Found G.S.Soln/ #GS
1	9(10) / 10	4(5) / 10	2/2
2	6(8) / 8	5(5) / 8	2/2
3	4 (5)/ 6	0(1) / 6	2/2
4	7(8) / 8	3(4) / 8	2/2
5	3(5) / 6	3(3) / 6	2/2
6	4(5) / 6	1(2) / 6	2/2
7	10(11) / 11	9(10) / 11	2/2
8	5(6) / 6	1(2) / 6	2/2
9	4(5) / 6	0(2) / 6	2/2
10	6(7) / 7	2(4) / 7	2/2
11	9(11) / 12	2(4) / 12	2/2
12	4(5) / 6	1(3) / 6	2/2
13	8(9) / 9	4(5) / 9	2/2
14	1(3) / 4	0(1) / 4	2/2
15	4(5) / 6	1(2) / 6	2/2
16	4(5) / 5	1(2) / 5	2/2
17	5 (7)/ 7	3(3) / 7	2/2
18	10(11) / 11	2(4) / 11	2/ 2
19	4(6) / 6	4(4) / 6	2/2
20	0(1) / 1	0(1) / 1	1/1
21	8(10) / 10	7(8) / 10	2/2
22	4(5) / 5	2(3) / 5	2/2
23	1(3) / 3	0(1) / 3	2/2
24	2(3) / 6	0(1) / 6	2/2
25	5(6) / 6	1(2) / 6	2/2

Table 20: 20x20 TPXOver Strongly Dominating Solution Counts, m04x06p50t100g2000

2.3.2. Nearly-Stable Matches

A stable match is one in which there are no pairs of matched couples that are mutually unstable. A nearly-stable match is one in which there are few pairs of matched couples that are mutually unstable. In a one-away match there is only one pair of matched couples that is mutually unstable. We emphasize that in such a one-away match there is no guarantee that swapping the unstable pair of couples will produce a stable match. The new couples resulting from the swap may be unstable with many other couples and unraveling may well be possible [14]. Our GA typically was able to find many one-away solutions and many of these were Pareto superior to the GS/DAA solutions. See Table 21 and Figure 2.3.2 for a graphical presentation.

Test Case	D1GS/TC	D2GS/TC
1	121 / 128	31 / 128
2	100 / 143	88 / 143
3	31 / 140	1 / 140
4	78 / 97	49 / 97
5	49 / 96	39 / 96
6	28 / 83	7 / 83
7	113 / 124	75 / 124
8	53 / 67	15 / 67
9	61 / 94	9 / 94
10	78 / 110	14 / 110
11	75 / 154	20 / 154
12	10 / 97	7 / 97
13	82 / 106	18 / 106
14	27 / 65	12 / 65
15	73 / 134	7 / 134
16	56 / 87	14 / 87
17	72 / 87	56 / 87
18	52 / 89	11 / 89
19	43 / 84	27 / 84
20	1 / 38	1 / 38
21	124 / 148	109 / 148
22	64 / 82	35 / 82
23	11 / 61	1 / 61
24	20 / 96	8 / 96
25	76 / 107	8 / 107

Table 21: 20x20 TPXOver 0.8, Mutation 0.3, One-Away Strongly Dominating Solution Counts

2.4. Extended Matching Problems - Hospitals Residents Problem with Couples (Couples Problem)

In the study of comparison between the stable marriage problem and the hospitals residents problem (a.k.a the couples problem) mentioned earlier, Aldershof and Carducci [1996] pro-

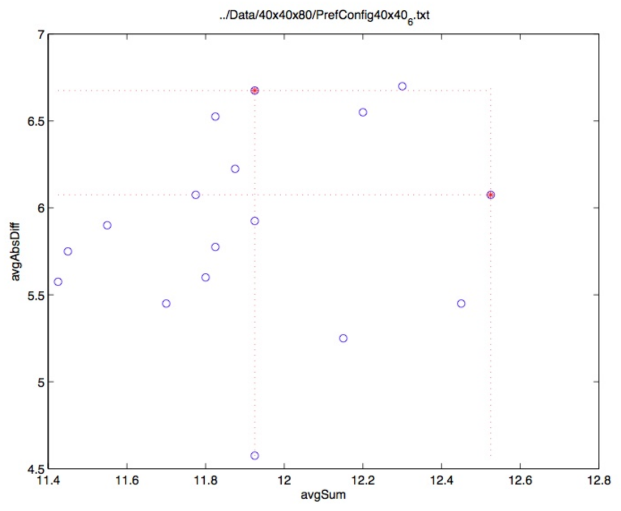


Figure 12: One-away solutions compared to GS/DSS solutions, 40×40 case6, see Table 22. GS/DAA in red ⋆.

Test Case	D1GS/TC	D2GS/TC
1	6 / 8	4 / 8
2	19 / 20	8 / 20
6	13 / 16	6 / 16

Table 22: 40x40 Case1,2,6 GA TPXOver Strongly Dominating Opt Solution Counts

vide an example which shows the possibility of different stable matchings yielding different sets of positions filled and of students assigned for a couples problem. Motivated by SFAS[2]'s decision to accommodate couples on 2009, Biro et al. [2011] illustrate how the SFAS algorithm, which is very similar to NRMP (Roth-Peranson) algorithm[3]) is unable to find an existing stable matching for a Hospitals Residents problem with 8 students, 8 positions and three pairs of couples. The proposed GAs solve both Aldershof's and Biro's Hospitals Residents (couples) problems to stable solutions. Before sharing the results of Biro's example, we present the model of the Couples Problem and the definition of Biro's problem.

[2]Scottish Foundation Allocation Scheme's (SFAS) is the Scottish equivalent of the NRMP.

[3]With the main distinction that agents, i.e., single applicants or couples, are introduced into the market one at a time, and after each such introduction, the resulting sequence of application, rejections, withdrawals, etc. is allowed to continue until stability is achieved before the next agent is introduced.

2.4.1. Model of Hospitals Residents Problem with Couples (Couples Problem)

Like all two-sided matching problems, the Couples Problem (Hospitals Residents Problem with Couples) consists of two finite and disjoint sets of agents, H and D, each of whom has complete and transitive preferences over the individuals in the other set. The follows are definitions of different components in a matching:
About hospitals:

- Let H be the set of hospitals participating in the matching.
- Let $\emptyset$ be the option of being unmatched. Define $\tilde{H} = H \cup \emptyset$. The existence of options outside of H indicates the possible pairing of an agent to nothing ('unmatched').

About doctors (residents):

- Let S be the set of single doctors.
- Let C be the set of couples of doctors. Let $c = (fm, sm)$ be any couple in the set C, where fm denote the first member of couple c and sm denote the second member of couple c. (fm_c, sm_c) refers to the members of a specific couple c. Let FM be the set of first members that form couples, i.e., $FM = \{fm|(fm, sm) \in C\ for\ some\ sm\}$. Let SM be the set of second members that form couples, i.e., $SM = \{sm|(fm, sm) \in C\ for\ some\ fm\}$.
- Then we have the set of doctors participating the matching as $D = S \cup FM \cup SM$.

About preferences:

- Single Doctor Preference: each single doctor $s \in S$ has a preference relation P_s over $\tilde{H}$. Let $h, h' \in \tilde{H}$. Preferences are assumed to be strict, i.e., if hP_sh' and $h'P_sh$, then $h = h'$. We say doctor s prefers h to h' if hP_sh' and $h \neq h'$. We say that hospital h is acceptable to single doctor s if $hP_S\emptyset$.

- Couple Doctor Preference: each couple $c \in C$ has a preference relation P_c over $\tilde{H} \times \tilde{H}$, pairs of hospitals (and possibly being unmatched). Preferences are assumed to be strict, as for single doctor preference. We consider the pair of hospitals (h, h') acceptable to couple c if $(h, h')P_c(\emptyset, \emptyset)$. We say that hospital h is listed by P_c if $\exists h' \in \tilde{H}$ s.t. either $(h, h')P_c(\emptyset, \emptyset)$ or $(h', h)P_c(\emptyset, \emptyset)$.

- Hospital Preference: each hospital $h \in H$ has a preference relation over all possible subsets of doctors, 2^D. Hospital preferences are assumed to be strict. Let κ_h be a positive number. We say that the hospital preference relation $\succeq_h$ is responsive with capacity κ_h if it ranks a doctor independently of her/his colleagues and prefers being 'unmatched' to any set of doctors exceeding capacity κ_h. Let P_h be the corresponding preference list of hospital h, which is the preference relation over individual doctors and $\emptyset$. We say hospital h prefers doctor d to d' if dP_hd' and $d \neq d'$. We say that doctor d is acceptable to hospital h if $dP_h\emptyset$. Let $\succeq_H = (\succeq_h)_{h \in H}$.

Now a matching market $\mathscr{M}$ can be referred to as a tuple $(H, S, C, (\succeq_h)_{h \in H}, (P_i)_{i \in S \cup C})$. A matching μ of the given matching market $\mathscr{M}$ is a function defined on the set $\tilde{H} \cup S \cup C$. μ specifies which doctors are paired to which hospitals (if any) such that $\mu(h) \subseteq D \; \forall \; h \in H$, $\mu(s) \in \tilde{H} \; \forall \; s \in S$, and $\mu(c) \in \tilde{H} \times \tilde{H} \; \forall \; c \in C$ where

- $\mu(s) = h$ if and only if $s \in \mu(h)$ and

- $\mu(c) = (h, h')$ if and only if $fm_c \in \mu(h)$ and $sm_c \in \mu(h')$

In order to define the stability of a matching μ for a given $\mathscr{M}$, $(H, S, C, (\succeq_h)_{h \in H}, (P_i)_{i \in S \cup C})$, we should first examine the hospital choices over acceptable sets of doctors. For any set $D' \subseteq D \cup C$, let $Ch_h(D')$ be the **choice** of hospital h given D' such that

- $Ch_h(D') \in \mathscr{A}(D')$,

- $Ch_h(D') \succeq_h D''$ for all $D'' \in \mathscr{A}(D')$, where

$$\begin{aligned}\mathscr{A}(D') = \{D'' \subseteq D | \forall\, s \in S, \qquad & if\ s \in D''\ then\ s \in D', \\ \forall\, c \in C, \qquad & if\ \{fm_c, sm_c\} \subseteq D''\ , then \{fm_c, sm_c\} \subseteq D', \\ & if\ fm_c \in D''\ and\ sm_c \notin D'',\ then\ fm_c \in D', \\ & if\ fm_c \notin D''\ and\ sm_c \in D''\ , then\ sm_c \in D'\}.\end{aligned}$$

Per the definition, $\mathscr{A}(D')$ is the collection of sets of doctors available for a hospital to employ when a set of doctors (D') is applying to it. This definition differentiates the applications by individual couple members and the ones by couples as a whole. For example, if $(fm, sm) \in D' \cap C$ but $fm \notin D'$ and $sm \notin D'$, then the hospital is acceptable to the couple if and only if both members of the couple are employed together. Whereas if $(fm, sm) \notin D'$ but $\{fm, sm\} \subseteq D'$, then the couple is happy to have one member matched to the hospital but not both.

The choice $Ch_h(D')$ is the most preferred subset of doctors among those in D' such that each couple is either matched or not matched to the hospital together if they apply as a couple. Since there is no couple for the simple matching problem, the set $\mathscr{A}(D')$ is simply the set of subsets of D'. Therefore, the choice $Ch_h(D')$ is the subset of D' that is most preferred by h (see Roth and Sotomayor [1990] for example).

Given that the choice $Ch_h(D')$ is the most preferred subset of doctors among those in D' which accommodates the couple applicants, we now list the possible cases where coalitions block a matching:

- A pair of one single doctor $s \in S$ and one hospital $h \in H$ blocks a matching μ if s prefers h to her/his currently paired hospital $\mu(s)$ and the hospital h prefers s to its currently paired doctor $\mu(h)$, i.e., if:
  - $hP_s\mu(s)$ and

  - $s \in Ch_h(\mu(h) \cup s)$

- A coalition of a couple $c \in C$ and two distinct hospitals $\{h, h'\} \in \tilde{H}$ blocks a matching μ if the couple c prefers the pair of hospital (h, h') to its currently paired hospitals $\mu(c)$ and hospital h and h' prefer the first and second member of c respectively, i.e. if:
  - $(h, h')P_c\mu(c)$,
  - $fm_c \in Ch_h(\mu(h) \cup fm_c)$, and
  - $sm_c \in Ch_h(\mu(h') \cup sm_c)$.
- A pair of one couple $c \in C$ and one hospital $h \in H$ blocks a matching μ if c prefers h to its currently paired hospital(s) $\mu(c)$ and h prefers both member of c to its currently paired doctors, i.e., if:
  - $(h, h)P_c\mu(c)$ and
  - $\{fm_c, sm_c\} \subseteq Ch_h(\mu(h) \cup c)$.

Besides the forming of blocking coalitions, it is also necessary to consider the possibility of participating agent(s) unilaterally rejecting a matching μ. A matching μ is said to be *individually rational* if no participating agent can be better off by unilaterally rejecting its currently paired partner.

Finally, a matching μ is deemed stable if there is no blocking of μ and it is individually rational.

2.4.2. Example Problem: the Special Hospitals Residents Problem with Couples (SHRC)

The example instance of SHRC consists of a set of residents (or applicants), a set of programmes (or hospitals), and a set of couples. Each of the agents (residents and programmes) has complete and transitive preferences over the individuals in the other set. Relationships among the participating agents (residents and programmes) are described as follows:

- About the participating agents:
    - All participating agents belong to the set A. A participating agent can be either a resident or a programme.
    - Each programme p offers a fixed number $c(p)$ of places, the *capacity* of the programme.
    - Each couple consists of a pair of distinct applicants.
    - No applicant can be in more than one couple.
    - An applicant is either *linked* or *single*, depending on whether or not the applicant is a member of a couple.
    - Each of applicant a and b is the partner of the other if a and b form a couple. i.e., $couple(a,b) \Rightarrow partner(a) = b \wedge partner(b) = a$.
- About the preferences:
    - Each applicant has a strictly ordered preference list containing a subset of the programmes (incomplete preference). i.e., $\forall\, a \in A$, $\exists\, PrefList_a$.
    - A programme that appears on the preference of an applicant a is said to be *acceptable* to a.
    - Applicant a is siad to prefer programme p to programme q if p precedes q in a's preference list. i.e., if $p \prec_{PrefList_a} q$ ($or\ rank_a(p) < rank_a(q)$), then $preference_a(p) > preference_a(q)$.
    - A pair programmes that appears on the joint preference list of couple(a,b) is said to be *acceptable* to that couple.
    - A couple(a,b) prefers a programme pair (p,q) to a programme pair (r,s) if (p,q) precedes (r,s) on (a,b)'s joint preference list, i.e., if $(p,q) \prec_{PrefList_{ab}} (r,s)$, then

$preference_{ab}(p, q) > preference_{ab}(r, s)$.

  - Each applicant a has a numerical score $s(a)$. Applicant a is *superior* to applicant b if $s(a) > s(b)$. All applicant scores are distinct.
  - The preference list of a programme is derived directly from the applicant scores. All programmes share the master preference list which consists of scores of all applicants.

- About a matching M: a matching M is a set of applicant-programme pairs satisfying the following conditions:
  - each applicant a appears in at most one applicant-programme pair.
  - either a couple(a,b) can be assigned to an acceptable pair of programmes (p, q) for (a, b) in M or there is no pair in M containing a or b.
  - the number of pairs in M containing the program p is at most $c(p)$.

The actual layout of the example problem is as follows: there are eight programmes, $p_1 \cdots p_8$, each with just one open position. There are eight applicants (residents) $a_1 \cdots a_8$, comprising three couples (a_1, a_5), (a_2, a_4), (a_6, a_8) together with two single applicants a_3 and a_7. The applicants are numbered in decreasing order of score (a_1 highest, a_8 lowest), and the individual and joint preference lists are as listed in Table 23.

2.4.3. Evolutionary Matching for SHRC

We modify the Stable Matching GA used for the simple marriage matching problem earlier. Major changes made are:

- Solution presentation: we use a position-based solution presentation. Each locus presents an open position (hospital). The number assigned to a given locus indicates the ID of the applicant (resident) who is matched with the given position. Solutions are presented as permutations of $1, \cdots n$, where n is the total number of applicants, with

	Programmes		
Applicants	Top Choice	Second Choice	Third Choice
a_1	p_1	p_3	
a_2	p_4	p_1	p_3
a_3	p_1	p_5	
a_4	p_5	p_2	p_7
a_5	p_2	p_6	
a_6	p_6		
a_7	p_6	p_8	
a_8	p_8		
(a_1, a_5)	(p_1, p_2)	(p_3, p_6)	
(a_2, a_4)	(p_4, p_5)	(p_1, p_2)	(p_3, p_7)
(a_6, a_8)	(p_6, p_8)		

Table 23: Biro Example 3 Preference Lists

a certain probability that each locus be reassigned to 0 for indicating the unassigned situation for the incomplete preference, e.g., a solution (2 4 1 0 3 5 0 7) shows that applicant 2 (resident 2) is assigned to position 1 while no one is assigned to position 4.

- Mutation and crossover operators are modified to accommodate the 'unassigned' (0) situation of a position. There could be zero, one, or multiple positions to be 'unassigned' in a given solution.

- Instability calculation: the fitness of a solution is still based on the instability of the solution, but the calculation of instability is modified in order to reflect the possible 'unassigned' status of an agent. The incomplete preference list indicates the possibility that a participating agent prefers to be matched to no one (unassigned) when no candidate on her/his/its preference list is available. Similarly, the hospital could prefer not to take in any resident if no one on its preference list is available. We introduce the utility score for the participating agents (applicants and positions) based on their preferences. The utility score of a position p for applicant a in the pairing (a, p) is defined as the distance between the lowest ranking and the ranking of position p on applicant a's preference list, i.e., $U_a(p) = max_{p \in PrefList_a} rank_a(p) - rank_a(p)$. For agent a the utility score 0 is assigned for pairing with no one, i.e., (a, '*unassigned*'). Pairing of agent a with any *unacceptable* agent from the other side generates a negative utility score. The instability between a pair of agents from different sides is decided by the potential improvement on the utility score of any of the member in the pair.

- Fitness evaluation: when a solution contains unacceptable pairing(s), a penalty is imposed for the occurrence of unacceptable pairing. The penalty reduces the fitness of the solution.

- Social welfare calculation: in order to taking into account the possible 'unacceptable' and 'unassigned' pairings, the social welfare calculation is modified to base on the utility score instead of the original preference rankings. The most preferred assignment gains the highest utility score while the 'unassigned' status gains zero utility score and the 'unacceptable' pairing gains negative utility score.

2.4.4. Experiment Results

The sequencing of applicants during the processing is an important matter for Roth-Peranson sequential couples algorithm (SCA), which is used by NRMP. Many applications use its variants and face the same problem: it could reach a certain matching and then cycle for the unsolvable sub-instance induced by some subset of applicants. For the given example, the SCA fails to converge to the awkward unique stable matching solution
$M = \{(a_1, p_3), (a_2, p_1), (a_3, p_5), (a_4, p_2), (a_5, p_6), (a_7, p_8)\}$
Nevertheless, as we expected, this is not an issue for GA. Parameters of the modified GA are set as follows: population size = 40, generation = 100, trial count = 10, mutation rate = 0.09, crossover rate = 0.25. All possible solutions with stability count less or equal to two are found by the modified Stable Marriage GA and listed in Table 24.

In Figure 13, we plot the candidate solutions with regard to three different pairs of objectives, social welfare vs. equity, social welfare vs. regret, and regret vs. equity. For both criteria 'equity' and 'regret', the lower the value the better a solution is. On the other hand,

						Objectives Combination		
						Sum-Regret	Sum-Diff	Reg-Diff
SID	Instability	Solntion	SumAvg	DiffAvg	RegretAvg	Dom. Soln	Pareto Frontier Soln	
1	0	2 4 1 0 3 5 0 7	19.2	5	16.9		v	v
2	1	1 5 2 0 3 0 4 7	19.2	7.33	17.6			
3	1	1 5 2 0 3 6 4 8	25.56	10	16.11	v	v	v
4	1	2 4 1 0 3 5 0 0	15.45	4	18.18		v	v
5	2	1 5 0 2 4 0 0 7	17.27	6.8	17.91			
6	2	1 5 0 2 4 6 0 8	22.8	10	16.6			
7	2	1 5 2 0 3 7 4 0	20	8.67	17.6			
8	2	3 0 1 2 4 5 0 7	21.6	6.67	16.2		v	v

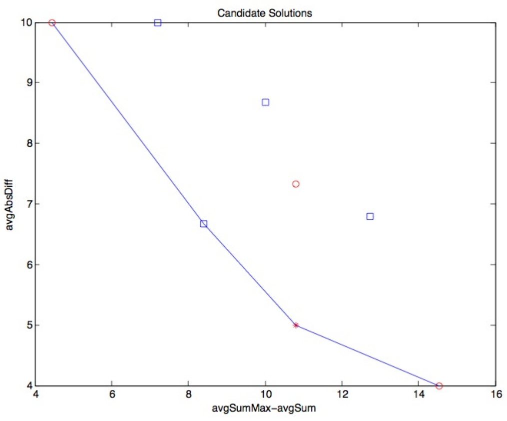
Table 24: Candidate Solutions for Biro Example 3 Problem (m009x025p40t10g100)

for the criterion 'social welfare', the higher the value the better a solution is. To make it easier on reading the graphics, when presenting the social welfare of a solution, we plot the candidate solutions with the difference between the maximum possible social welfare score and the social welfare score of the given solution. By doing so, the smaller the difference, the better the solution is.

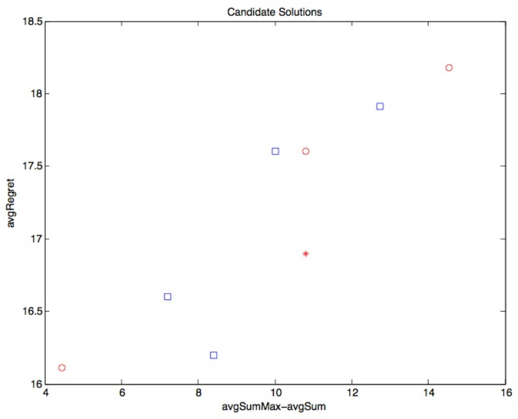
Three groups of candidate solutions, in terms of solution stability, are shown in Figure 13. The red star indicates the one and only stable solution, which is the solution with a zero instability count. The red circles indicate the solutions with instability count that equals one, and the blue squares indicate the solutions with instability count that equals two. We identify the same four solutions (#1, #3, #4, #8) which form the Pareto Frontier for two pairs of objectives: 'social welfare vs equity' and 'regret vs equity'. From the plot of objective pair 'social welfare' vs. 'regret', we see the candidate solutions are more or less along the same trend line, with the solution #3 dominates the others. Moving along the Pareto Frontier, we have the examples of possible trading off among different objectives with different candidate solutions.

2.5. Summary And Discussion

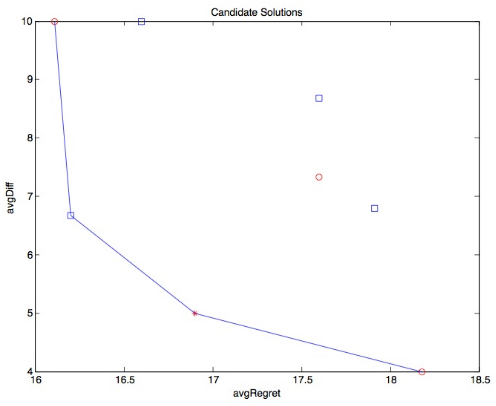
The stable marriage problem is prototypical of the two-sided matching problem in which agents who have preferences, interests and capacities to act on their own are paired up or matched. Standardly, 'stability' is the major concern when trying to match the participating agents, and variants of the well-known Gale-Shapley deferred acceptance algorithm



(a) Objectives: Social Welfare & Fairness



(b) Objectives: Social Welfare & Regret



(c) Objectives: Regret & Fairness

Figure 13: Candidate Solutions for Biro Example 3 with Different Paired Objectives

(GS/DAA) are used to find stable matches. In this chapter, we question the emphasis on absolute stability for a matching problem, we discuss the two key properties of GS/DAA, and we investigate the stable marriage matching problem as a multi-objective problem via the evolutionary computation approach. We state the rationale and summarize our findings as follows:

- Conventionally, matches are evaluated in terms of 'stability'. A match is said to be stable if there is no pair of matched agents in it containing individuals who prefer to be matched to each other but are not. The concern for an unstable matching solutions is that the unstable pair will abandon the current assigned partners, break at least one other pair of partnership, in order to form new pairs according to the agents' preference. Therefore, requiring matches to be stable in the first place will prevent breakup and reformation among pairs and its attendant costs. But, in reality, we note that in many cases it may be practically difficult (or made practically difficult by the operator of the centralized market) for members of matched couple to undertake swaps, regardless of their preferences. To be more specific, when the number of unstable pairs is small but non-zero, there will in many cases be no realistic means for the pairs to find each other and unravel the matching. This gives the decision maker some leeway with regard to a solution's stability and raises the issue that social welfare, equity, and whatever other measures of performance are relevant should be taken into consideration too, since decision makers, including agents participating in a centralized market, may quite reasonably want to exchange some stability for improvements in, say, social welfare or equity. This makes a strong case for insisting that two-sided matching be viewed as a multi-objective problem.

- Under the special assumptions, the stable marriage problem and the admission problem have stable matches and the DAA can find one and will find one quickly. Unfortunately, when going beyond the simple matching problem, the optimal stable matches do not exist in general. It has been shown that, with the further generalized matching problems, the DAA approach and its likes fail to find the existing stable solutions.

- Asymmetry is a general characteristic of the DAA style approaches. In the DAA process, one side of agents proposes, the other disposes. Focusing now on the marriage problem, if the men propose, they obtain a stable match that is male optimal in the sense that no man in this match strictly prefers (does better in) any other stable match. Conversely, the match is female pessimal in the sense that no woman is worse off in any other stable match. And vice versa if the women propose. It raises the important question in terms of both equity and social welfare, whether better matches exist and can be found via other methods.

- With the two-population GA approach, we show that:

  - Typically in simple marriage matching problems there are stable matches that are Pareto superior to the deferred acceptance matches, in regard to equity and social welfare (as we have characterized them).

  - Also there are many nearly-stable matches that are superior to the deferred acceptance style matches (stable solutions) on equity and/or social welfare.

  - For the more generalized matching problem, GA offers the existing stable solution which the DAA style approaches fail to find.

  These make a strong case for one to look well beyond the GS/DAA style solutions. Particularly, heuristic alternatives to to deferred acceptance may well be able to offer practical improvements and complements to deferred acceptance in the centralized markets

- Scale is an important issue. As the problem size exceeds 50, the GA generally continues to perform well, but this needs more extensive testing. We note that these findings apply to the randomly-generated test problems we have used. In these problems, preferences are uncorrelated. Actual applications may be different (think of the marriage market). When preferences are identical there will be only one stable match and, generally, the number of stable matches will decline with increasing correlation of preferences. The effects on nearly-stable matches are not known to us; empirically they remain abundant.

CHAPTER 3 : Finding Robust Solutions with the Metaheuristic Approach

3.1. Introduction

In this chapter, we propose and explore, in the context of benchmark problems for flowshop scheduling, a risk-based concept of robustness for optimization problems. This risk-based concept is in distinction to the uncertainty-based concept employed in the field known as robust optimization. Most importantly, our approach provides probabilistic support for the decision-maker with potential solutions that are not otherwise readily available. To further examine the risk-based robust optimization approach, we propose a new uncertainty-based scheme that shares the same core procedure of our risk-based approach. We demonstrate the new scheme by solving different one- and two-variable functions with broad and sharp peaks.

Structure of the chapter. In Section 2.2, we present the different approaches for robust optimization from the literature, and we present the motivation for our study. In Section 3.3, we propose the risk-based approach and explore our scheme in the context of benchmark problems for flowshop scheduling. To answer the question raised from our experiment results, we suggest a new uncertainty-based scheme and examine it with different test functions in Section 3.4. Finally, Section 3.5 contains our concluding remarks.

3.2. Motivation and Related Work

Many optimization research efforts are built upon the hypothesis of perfect information, i.e., accurate values for the system parameters and exact knowledge of the random variables. Assuming that the input data are precisely known and equal to some nominal values, these studies develop models and offer effective methods to solve different NP-hard optimization problems. While these proposed approaches work well for the deterministic (nominal) versions of the studied problems, they often do not take data uncertainty into account. Without knowing the influence of data uncertainties on the quality and feasibility of the model, it is unclear how these proposed methods will perform when we apply them to solve the same optimization problems with less-than-perfect information, which we tend to have for most real-life problems. The so-called 'optimal solutions' found for the nominal problems may no longer be satisfactory. Regarding solution quality, we are concerned that the 'optimal solutions' may not perform equally well when the data takes values different than the nominal ones. Regarding solution feasibility, we worry that the 'optimal solutions' may become infeasible even if the nominal data are only slightly perturbed in realization, especially when the data uncertainty is associated with critical constraints. The need to design solution approaches that are immune to data uncertainty prompts and results in the vibrant research field of robust optimization.

To put it simply, something is robust if it performs well under varying conditions. Wagner's characterization is representative and applies to any system: "A biological system is robust if it continues to function in the face of perturbations" [Wagner, 2005, p.1], although often we will want to add "well" after "function" (see also Kirschner and Gerhart [1998]). As noisy, incomplete or erroneous data exist everywhere, the general notion of robustness is apt for and useful in many fields, including biology, engineering, finance, and management science. Numerous studies give examples of the need of robustness in searching for problem solutions (Morgenstern [1963], Goldfarb and Iyengar [2003], Bertsimas and Thiele [2006b]).

The concepts of robustness and robust design optimization have been developed independently mainly in the fields of operations research and engineering design. In engineering design, the early attempts to account for design uncertainties in the framework of quality engineering are due to Taguchi's three-stage robust design methodology (Taguchi [1984, 1986]). In operations research, the linear model that constructs a feasible solution for all data belonging in a convex set was first proposed by Soyster [1973] and further developed by Falk [1976] and Singh [1982]. The notion of robust optimization gained focus in OR after the publication of Mulvey et al. [1995]. Arguing that the optimization-based system on the whole fails to address risk aversion as specified in classical decision theory, Bai et al. [1997]

suggest that a concave utility function should be incorporated in a model for risk-averse decision-making . Exploring robustness as performance in the worst-case condition(s), a large literature characterizes a robust solution to an optimization problem as one that is optimal in the worst case, and then seeks effective methods for discovering such solutions (see Ghaoui and Lebret [1997], Bai et al. [1997], Ben-Tal and Nemirovski [1998, 2000, 2002], Beyer and Sendhoff [2007]). Examining the over-conservativeness issue, Bertsimas and Sim [2004], Bertsimas and Thiele [2006a] propose approaches that flexibly adjust the level of conservatism and protect against violation on constraints deterministically .

Also known as maximin (maximizing the minimum possible return), "worst-case optimal" is the solution principle for games of pure conflict. Maximin is arguably often a reasonable principle to invoke when decision-making occurs under uncertainty, that is, when it is not possible or credible to assign probability distributions to the relevant conditions. This observation suggests the possibility of defining a robust-under-risk notion of robustness, one that is appropriate for situations in which probability distributions are available for relevant aspects of the context. The main aim of this chapter is to explore this suggestion. Before going into the details of our setup, however, other related work must be mentioned.

Scheduling problems generally (Cheng et al. [1996, 1999], Pinedo [2008]) and flowshop problems in particular have long been addressed, and addressed successfully, with evolutionary algorithms (Hart et al. [2005], Hejazi and Saghafian. [2005], Reeves [1995]). For the most part, the focus has been on finding heuristically optimal solutions, rather than robust solutions, however robustness may be defined. Exceptions are Herrmann [1999] and Jensen [2003] (see also comment in Beyer and Sendhoff [2007]). Herrmann's work assumes a minimax (worst-case-optimal) notion of robustness. Using what is now called a teacher-learner coevolution framework, Herrmann (Ficici [2008]) evolves two populations together, one of scenarios (parameterizations of the optimization problem, the "teachers" in modern parlance) and one of the solutions (the "learners"). Heuristically, evolution of the teachers finds the most unfavorable scenarios and evolution of the learners finds the best solutions for the worst-case scenarios. Jensen defines (as a heuristic) the robustness of a solution S to be the weighted average makespan of solutions in the neighborhood of S, then uses a genetic algorithm to find robust solutions so characterized. Both of these approaches are quite different from ours. In future work it will be important to compare and contrast the results of all three methods.

3.3. Robust-under-Risk

We propose a robustness-under-risk score, R_R, based on four elements: the problem or model (M), the solution (S), the perturbation regime (P), and the level of certainty ($L, 0 < L < 1$). The robustness of a solution S to model M under perturbation regime P with the level of certainty L is denoted as $R_R(S, M, P, L)$. Our proposal is to measure the robustness of a solution (to a model, $\cdots$) as the best objective value for which the probability is at least L that the solution will match or improve on it. A *robust solution* is a solution whose robustness-under-risk score is optimal, that is, better than or equal to the robustness-under-risk scores of all other solutions. In what follows, we explore how these quantities can be estimated by collecting high-quality solutions that appear during the normal runs of evolutionary computation.

3.3.1. Description of the Proposed Approach

Assuming that we are solving a minimization problem and using perturbation regime P to represent the ambient risk, then we can estimate the robustness of a solution S as follows:

1. Using P, generate a sample of perturbed problems, $PSample$.

2. Obtain the objective function value of solution S on each member of the sample set $PSample$.

3. Sort the objective values so obtained in ascending order (more generally from best to worst).

4. Estimate the robustness of S at level L to be the objective function value of S at the

L^{th} decile of the sample $PSample$. (Objective function values above the L^{th} decile are all inferior to every value at or below the L^{th} decile.)

Note that so conceived, the robustness score of a given solution by itself tells us nothing about whether the solution is optimal or even good with respect to robustness. To evaluate a solution in this regard we must somehow compare it to other solutions and their robustness scores. Our proposal is to obtain heuristically a sample of high-quality solutions (which we call the solutions of interest, or $SoIs$, for this problem, see Kimbrough et al. [2010]) and to estimate robust solutions as best available in the SoIs.

This method brings us to the question of how to discover robust solutions to an optimization problem. Given our proposed robustness measure, an obvious approach would be as follows:

1. Using P, generate a large sample of perturbed problems, $PSample$.
2. Solve each problem in $PSample$ to optimality. The collection of optimal solutions constitutes the SoIs, the solutions of interest, from which comparatively robust solutions are to be sought.
3. Using P, generate a new large sample of perturbed problems, $PSample'$.
4. Obtain the objective function value of each solution in SoIs for each member of the sample $PSample'$.
5. Sort the objective values so obtained in ascending order (more generally from best to worst).
6. Estimate the robustness of $S \in$ SoIs at level L to be the objective function value at the L^{th} decile of the sample $PSample'$. Evaluate every member of SoIs in this way.
7. Designate as robust any member of SoIs with a best estimated robustness score.

We would find this a potentially reasonable procedure except for step (2). While the matter certainly deserves empirical investigation, the computational cost will often be prohibitive.

Further, there would seem to be little reason *a priori* for an optimal solution to one perturbed problem to be robust for another perturbation.

We explore a different approach, one that we think needs to be investigated in any case. In this approach to the discovery problem, we make use of the fact that in evolutionary computation (as well as other metaheuristics) large numbers of solutions are considered during runs of the algorithm(s). We thus propose to replace step (2) with step (2'):

2'. For each problem in *PSample*, obtain a sample of distinct high-quality solutions. The union of these solutions constitutes the SoIs, the solutions of interest, from which comparatively robust solution are to be sought.

In what follows we report on a series of experiments in finding robust solutions to flowshop scheduling problems using the general approach discussed. We begin in the next section with a precise description of our setup.

3.3.2. Experimental Setup for Robustness-Under-Risk

Flowshop GA

In a flowshop problem we are given m machines and n jobs to be scheduled for processing one at a time on the machines. We are also given a processing time array: $Processing(i, j)$ = processing time of job j on machine i. The problem is to find a schedule (a permutation of the jobs) with the minimal makespan (time to complete all jobs). The measure of performance (fitness) for a solution is its makespan. For further details on flowshop problems, see Pinedo [2008] and other references cited above. Figure 14 gives the algorithm in pseudocode for calculating makespans.

In our study, we experimented with fourteen flowshop scheduling problems that are by now standard benchmarks: ISHardest (Ignall and Schrage [1965]), Car1, Car2, Car3, and Car4 (Carlier [1978]), and Txxxx (Taillard). Table 25 lists the problem sizes and in the fourth column, shows how close our rather conventional genetic algorithm came to optimally

```
1. Given:
   (a) $m$ machines; $n$ jobs;
   (b) $Decision_{n\times 1}$, a permutation vector of job IDs, $1 \cdots n$;
   (c) $Processing_{m\times n}$, array of processing times, $Processing(i,j)$ = processing
       time of job $j$ on machine $i$;
   (d) $Start_{m\times n}$, array of starting times, $Start(i,j)$ = starting time of the $j^{th}$
       job from $Decision$ ($Decision(j)$) on machine $i$, initialized to 0;
   (e) $Completion_{m\times n}$, array of completion times, $Completion(i,j)$ = comple-
       tion time of the $j^{th}$ job from $Decision$ ($Decision(j)$) on machine $i$, ini-
       tialized to 0;
   (f) $Availability_{m\times 1}$, array of next availability times for machines,
       $Availability(i)$ = next available starting time for machine $i$, initialized to
       0;
2. For $j = 1$ to $n$:
   (a) For $i = 1$ to $m$:
        i. $Start(i,j) = Availability(i)$
       ii. $Completion(i,j) = Start(i,j) + Processing(i, Decision(j))$
      iii. $Availability(i) = Completion(i,j)$
3. $Makespan = Completion(m,n)$
```

Figure 14: Makespan calculation procedure, for standard simple flowshop problems

solving the problems.

Solution Representation

For the flowshop benchmark problems, we use the job-based representation. This is perhaps the most natural representation of a schedule. The chromosome is a string of job IDs. Different job ID permutations present different solutions. The string length depends on the total number of jobs (n) to be completed. For example, for problem ISHardest, which is a problem with 3 machines and 10 jobs, a chromosome for ISHardest may look like (3 7 8 5 4 2 6 10 1 9).

Test Problem	# Machines	# Jobs	ObjValDelta
ISHardest	3	10	0%
Car1	5	11	0%
Car2	4	13	0%
Car3	5	12	0%
Car4	4	14	0%
T1378	10	20	<0.073%
T1397	10	20	<0.073%
T1484	10	20	<0.135%
T1496	10	20	<0.869%
T1538	10	20	<0.520%
T1582	10	20	<0.127%
T1591	10	20	<0.440%
T1593	10	20	<0.252%
T1659	10	20	<0.724%

Table 25: Information on Benchmark Flowshop Problems

Population Initialization

Given population size (*popSize*), we first create a population of *popSize* chromosomes, where each chromosome is a random permutation of n job IDs (i.e., number $1 \cdots n$).

Evaluation function

The makespan calculation procedure will serve as the evaluation function since the measure of performance (fitness) for a solution is its makespan.

Genetic operators

Two classical genetic operators, mutation and crossover, are used during the alteration phase of our genetic algorithm:

- mutation: randomly picking two jobs on a solution and swapping them. The mutation rate is the probability that one pair of jobs is swapped within a chromosome. For example, a solution for ISHardest problem may be

(3 7 8 5 4 2 6 10 1 9).

The resulted chromosome after a mutation happened on the 3rd and 6th jobs will be

(3 7 $\underline{2}$ 5 4 $\underline{8}$ 6 10 1 9).

- crossover: we use *order crossover*, OX, as our crossover operator [Michalewicz, 1995, p.217]. OX builds offspring by choosing a subsequence of jobs from one parent chromosome and preserving the relative order of the rest jobs from the other parent. For example, assume that the cut points are randomly selected after the 3rd and 6th gene (and marked by "|"), the following two sample chromosomes

  p_1 = (1 2 3 | 4 5 6 | 7 8 9 10) and
  p_2 = (4 5 2| 1 8 7 | 6 9 3 10)

  would produce the offspring in the following way:

  1. The subsequence of jobs will be copied into offspring:

     o_1 = (x x x | 4 5 6 | x x x x) and
     o_2 = (x x x | 1 8 7 | x x x x).

  2. starting from the second cut point of one parent, the jobs from the other parent are copied in the same order, omitting symbols already present. When reaching the end of the chromosome, we continue from the first place of the chromosome. The sequence of the jobs in p_2 from the second cut point (starting from the 7th job) is

     6 9 3 10 4 5 2 1 8 7

3. after removal of the jobs 4, 5, and 6, which are already in the first offspring o_1, we get

   9 3 10 2 1 8 7

4. the resulted sequence is placed in o_1, starting form the second cut point:

   $\rightarrow o_1 =$ (x x x | 4 5 6 | 9 3 10 2)

   $\rightarrow o_1 =$ (1 8 7 | 4 5 6 | 9 3 10 2)

5. Similarly we get the second offspring o_2:

   $o_2 =$ (4 5 6 | 1 8 7 | 9 10 2 3)

Creating a new generation

Two more operators, *selection* and *elimination*, are used together with *mutation* and *crossover* during the alteration phase of our genetic algorithm:

- selection: using tournament-2 selection, two parent solutions are selected as mating candidates. Tournament-2 selection is implemented by randomly selecting two individuals from the current population, the individual with higher fitness value is the winner and will be used for the mating process. To generate one pair of candidate solutions, we need to perform tournament-2 selection twice.

- mating (mutation and crossover): for a pair of candidate solutions, each solution goes through mutation (with the given mutation probability p_m) individually first. The candidates are then mated (cross-overed) with a given crossover probability, p_x.

- elimination: The two parent candidates may or may not actually mutate or mate (crossover). In the case that no actual mating happens, the mutated parents (in some cases the parents may not even be mutated because crossover is not performed universally) will be kept in a temporary pool. In the case that the mating event actually happens, the two generated children solutions will be kept in the temporary pool. Together with the two original parent candidates, there are a total of four solutions in the temporary pool. The two solutions with lower fitness values will be eliminated.

- building the new generation: the survivors of the elimination process (the two solutions with higher fitness values) will be added into the next generation population.

The selection, mating, elimination, and building the new generation processes are repeated till we collect $popSize$, n, solutions for the new generation.

Solution Sampling

There are two conditions under which we used the GA to explore the solution space for each problem. Condition one is the "perturbed scenario"(PS), and condition two is "base scenario" (BS). Under the base scenario, the GA was directed to find good solutions to the original problem. Under the perturbed scenario, the elements in the processing times array, $Processing_{m\times n}$, were randomly subject to change, producing a perturbed processing times array $PProcessing_{m\times n}$:

$$PProcessing(i,j) =$$

$$\begin{cases} N(0, f \times Processing(i,j)) + Processing(i,j) & if\, N(0, f \times Processing(i,j)) > 0, \\ Processing(i,j) & otherwise; \end{cases}$$

where $N(\mu, \sigma)$ is a normal random deviate with mean μ and standard deviation σ, and f is the perturbation factor, which we set to 0.1. (Our results were not sensitive to the perturbation factor value of 0.1)

Phase I of our experiments proceeds as follows for a given flowshop problem:

1. For the PS, we randomly create $N_{pert} = 100$ perturbed processing time arrays: $PProcessing^1, \cdots, PProcessing^n$.

2. Initialize the perturbed solutions of interest, $PSoIs^1, \cdots, PSoIs^n$, as empty heaps, each with maximum size of $h = 300$.

3. For each of the n perturbed processing times arrays we conduct $r = 20$ independent runs of the standard GA, saving the $N_{topSoln}$ unique best solutions over the r runs in the associated $PSoIs^k$ heap. (In a seperate data structure we record how many times a solution in the heap is encountered.) The individual $PSoIs^k$are then combined into $PSoISH$(perturbed solutions of interest super-heap) by taking their union.

An analogous process is followed for the BS, except that the original processing time array is used throughout. This results in $USoISH$(unperturbed solutions of interest super-heap), as a result of r runs conducted n times on the original problem. This completes phase I - collecting the solutions of interest.

In Phase II, for a given problem, we generate 100 new perturbed processing time arrays, each for $f = 0.1$, 0.25, 0.5, 0.75, 1, 1.25, and 1.5. We evaluate every member of $PSoISH$ and $USoISH$ with every new perturbed processing times array. We then compare the performances of the two super-heaps, $PSoISH$ and $USoISH$.

Parameters

For the benchmark problems, we use the following parameters:

- *pop_size*: population size, *pop_size* $= 100$
- p_m: probability of mutation, $p_m = 0.1$
- p_x: probability of crossover, $p_x = 0.5$
- g : number of generation, $g = 500$
- t: number of trial, $t = 10$. Each trial starts with one randomly initialized population, and evolves through g generations.

- $N_{topSoln}$: number of recorded top solutions for each solved problem, $N_{topSoln} = 300$
- N_{pert}: number of perturbed problems, $N_{pert} = 100$
- f: perturbation factor, percentage of the original processing times used as the standard deviation of the assumed normal distribution for the environment noise to be added on the original processing times. f ranges between 0.1 and 1.5.

These values were arrived at by a mild tuning exercise on a sample of the test problems and then confirmed by tests on the remaining problems. The tuning is only approximate.

3.3.3. Experiment Results for Robustness-Under-Risk

We present our results by first discussing a representative problem among our 14 benchmark cases: the Taillard benchmark problem with 10 machines, 20 jobs, and best-known makespan of 1378. The results come in two forms: first, we compare the two super-heaps overall by assessing all of the solutions for each of the seven levels of f. (In the case of Taillard 1378, there were 29,993 distinct solutions in $PSoISH$ and 29,948 in $USoISH$.) Then, for each of the 100 problems at each level, we compare the objective function pairs for the robust solution in each heap, using a Wilcoxon test with the null hypothesis of no difference between the two solutions. Table 26 shows our results for the Taillard 1378 problem. The null hypothesis is only rejected for one level of f and the BS super-heap is even better. Considering the Bonferroni issues, with our seven tests and a significance level of 0.05, the probability of observing at least one significant result by chance is:

$$\begin{aligned} Pr(\exists\ significant\ result) &= 1 - Pr(no\ significant\ results) \\ &= 1 - (1 - 0.95)^7 \\ &= 0.30166 \end{aligned}$$

Taking the Bonferroni correction will set the significance cut-off at 0.05/7 = 0.00714286. With the new cut-off, $Pr(\exists\ significant\ result) \approx 0.0489$, which is just under our desired

0.05 level. Therefore, we do not claim that there is evidence that the BS actually produces significantly better solutions. The key point overall is that the solutions for the two scenarios are broadly equivalent in terms of performance.

f	p-value	reject null?	z-value
0.10	0.0094951	1	2.3457
0.25	0.15384	0	1.0201
0.50	0.63981	0	-0.35796
0.75	0.53456	0	-0.086741
1.00	0.50439	0	-0.010995
1.25	0.52776	0	-0.069637
1.50	0.54862	0	-0.12217

Table 26: Wilcoxon test results for super-heap comparisons on benchmark problem Taillard 1378

In the second form of our results, we obtain for each of the two super-heaps the estimated robust solutions at each level of f and for deciles $L = 0.99$, 0.95, 0.90, 0.85, and 0.80. Table 27 presents the results for the Taillard 1378 problem. Remarkably, the two super-heaps, and hence the two distinct GA approaches (perturbed and unperturbed), give very similar results. Again, we see that the BS solutions at $f = 0.1$ come out better at most values of L than the PS solutions, even though the PS solutions were produced from processing arrays generated identically.

As for the other 13 benchmark problems, our results are essentially in line with those for Taillard 1378. It is remarkable that there is so little difference with respect to robustness in the solutions in our two super-heaps. The best solutions found by the GA on the unperturbed problem are as good as, or perhaps better than, the best solutions found by the GA when trained on perturbed problems generated by the same process as the comparison test problems. This is not because the perturbations do not affect the makespans. As the entries in Table 27 (and in the corresponding tables for the other problems) show, makespans actually increase with increasing perturbations. A GA is (at least sometimes) itself a procedure for finding robust solutions.

Scenario	$PSize$	$L = 1.00$	$L = 0.99$	$L = 0.95$	$L = 0.90$	$L = 0.85$	$L = 0.80$
PS	0.10	1491.8	1489.2	1479.9	1471.9	1469.1	1466.9
BS	0.10	1492.1	1484.1	1472.6	1463.0	1460.2	1456.6
PS	0.25	1654.8	1645.2	1629.2	1624.1	1612.6	1608.1
BS	0.25	1652.0	1646.5	1629.7	1617.5	1611.8	1602.6
PS	0.50	1971.7	1958.3	1918.0	1884.7	1866.7	1858.8
BS	0.50	1971.7	1954.5	1909.3	1887.1	1869.3	1858.7
PS	0.75	2261.7	2244.5	2196.5	2167.3	2142.6	2120.7
BS	0.75	2263.7	2246.6	2204.8	2168.4	2139.4	2113.3
PS	1.00	2519.0	2512.5	2456.7	2399.7	2381.7	2357.8
BS	1.00	2522.9	2505.8	2448.9	2409.1	2369.9	2360.3
PS	1.25	2832.2	2824.0	2769.8	2715.6	2683.2	2647.7
BS	1.25	2824.4	2810.9	2759.5	2712.0	2673.9	2644.7
PS	1.50	3140.0	3119.2	3072.9	2998.4	2953.5	2914.5
BS	1.50	3192.7	3142.7	3065.2	3012.8	2936.8	2899.6

Table 27: Taillard benchmark flowshop problem. 10 machines. 20 jobs. 1378 minimum makespan.

3.4. Robust-under-Uncertainty

To answer the question raised from our previous experiment results, about whether GA itself is a procedure for finding robust solutions, we further propose a robustness-under-uncertainty score, R_U, based on five elements: the problem or model (M), the solution (S), the perturbation regime (P), the Level of rejection on suboptimal solutions (L) (e.g., rejection level $L = 0.90$ means to accept only the solutions with top 10% objective values), and the sampling frequency (F) ($0 < F$). The robustness of a solution S to model M under perturbation regime P with a given rejection level L and sampling frequency F is denoted as $R_U(S, M, P, L, F)$. Our proposal is to measure the robustness of a solution (to a model, $\cdots$), as the number of perturbed cases in which the solution (S) appears to be one of the solutions of interest for that specific case. Given our proposed robustness measure, an obvious approach would be as follows:

1. Using P, generate a large sample of perturbed problems, $PSample$, $F = size(PSample)$.

2. Solve each problem in $PSample$ to optimality.

3. With the same rejection level (L), for each problem in $PSample$, maintain an independent set of acceptable solutions , $ASoIs_i$, $i = 1 \cdots size(PSample)$. The collection of solutions from every $ASoIs_i$ constitutes the SoIs, the solutions of interest, from which comparatively robust solutions are to be sought.

4. For each solution in SoIs, tally the number of independent sets $ASoIs_i$ in which the solution appears. Estimate the robustness score of $S \in$ SoIs at acceptance level L as the independent set $ASoIs_i$ count.

Since "perturbation sampling" is the key point of our proposed approach, we will look at perturbation from a slightly different angle in order to simplify the process and focus on the effect of "perturbation sampling".

In evolution theory, the organism's physical properties directly determine its chances of survival and reproductive output. The inherited genes (the genotype) cause a trait, and a physical property (the phenotype) is the observable expression of the genes. While an organism's gene is a major influencing factor in the development of its corresponding physical property, there are other factors that could also influence the development of its phenotypic features. The level of phenotypic plasticity describes the strength of environmental influence on the particular phenotype that develops. During the development of its phenotype, a chromosome could interact with the environment and the resulting phenotype may vary. We consider such interaction between a chromosome and its environment as a perturbation of the development process, e.g., abnormal weather condition, individual nutritional imbalance, etc. The individuals with the "best" genes but that are sensitive to the environmental perturbation may not survive, since the resulting physical properties reduce their fitness. On the contrary, the individuals who have "good" genes and are robust to the perturbation have a better chance of survival in a noisy environment.

To take the existence of environmental noise into consideration, Fitzpatrick, Miller, et al. discuss different approaches for calculating the fitness values (see Fitzpatrick and Grefenstette [1988], Miller and Goldberg [1996]). These studies treat the environment noise by adding it to the fitness function. That is, if $x = (x_1, x_2, \cdots, x_m)$ is a phenotypic vector, with fitness evaluation function f and noise δ, the fitness of individual x will be $f(x) + \delta$. On the other hand, since the perturbation can be seen as the interaction between the chromosome and its environment during the development of its phenotypic feature, we can shift the noise into the genotype-phenotype development process by adding the noise $\Delta = (\delta_1, \delta_2, \cdots, \delta_m)$ directly to the phenotypic vector x. In this case, the fitness evaluation of x becomes $f(x+\Delta)$ and the solutions thus determined are expected to be more robust against the perturbations.

3.4.1. Description of the Proposed Approach

After shifting the noise from the problem parameters to the decision variables, we take the similar steps to generate high quality candidate solutions, as we have done previously. A sample of noise $DSample = (\Delta_1, \Delta_2, \cdots, \Delta_k)$ is generated. For each random noise Δ_i, the problem P is solved to optimality by using $f(x + \Delta_i)$ as its fitness evaluation function. The solutions of interest (SoIs) are collected in a corresponding heap $SoIsHeap_i$, $i = 1 \cdots k$. While taking the collection (union) of the k heaps, for each of the candidate solution in the final collection, we also keep track of the number of heaps in which the corresponding solution appears. The frequency of a solution's appearance in the different heaps is taken as its robustness-under-uncertainty score for this problem. Again, the robustness score of a given solution by itself tells us nothing about whether the solution is optimal or even good with respect to robustness. To evaluate a solution in this regard, we must somehow compare it to other solutions and their robustness scores. Our proposal is to obtain heuristically a sample of high-quality solutions (the $SoIs$) and to estimate robust solutions as the best available in the SoIs. The pseudocode for finding robust solutions with robustness-under-uncertainty measurement is described as in Fig. 15.

```
$for\ dsample\_count = 1..k:$
{    $Randomly\ generate\ \Delta_{dsample\_count};$
     $Randomly\ initialized\ population\ Pop_{gen\_count} = \{x^1, x^2, \cdots, x^{popSize}\};$
     $for\ gen\_count = 1 : numGen :$
     {    $for i = 1..popSize :$
          {    $y^i = x^i + \Delta_{dsample\_count};$
               $take\ f(y^i)\ as\ the\ fitness\ value\ of\ x^i;$
          }
          $Select\ Pop_{gen\_count+1}\ from\ Pop_{gen\_count}\ based\ on\ f(y^i);$
          $Apply\ genetic\ operators\ to\ each\ individuals\ in\ the\ population;$
          $Keep\ the\ top\ n\ solutions\ in\ SoIsHeap_{dsample\_count};$
     }
}
$Candidate\ solution\ set\ (SoIs)\ =$
    $all\ distinct\ solutions\ in\ all\ k\ SolsHeap_{dsample\_count};$

$\forall\ s \in SoIs :$
     $heapCount_s\ =$
          $the\ number\ of different\ SoIsHeap_{dsample\_count}\ where\ s\ appears;$
     $Take\ heapCount_s\ as\ R_U\ (robustness\text{-}under\text{-}uncertainty\ score)\ for\ solution\ s.$
```

Figure 15: Proposed robustness measure: robustness-under-uncertainty

3.4.2. Mathematical Model

The theoretical foundations of genetic algorithms rely on a binary string representation of solutions and on the notion of a schema. In this section, we will describe the simple mathematical model of our approach following the formula of Michalewicz [1995] and Tsutsui and Ghosh [1997].

First, without losing any generality, we can assume maximization problems only. Consider the schema theorem of the simple GA using a proportional payoff selection scheme and a single-point crossover. With the binary string representation, a schema is built by introducing the *don't care* symbol (*) into the alphabet of genes. A schema is a template presenting a subset of the search space; it represents all strings which match it on all positions other than '*'. For example, the schema, $S_e = (11 * * 0)$ matches four strings {(1 1 0 0 0), (1 1 0 1 0), (1 1 1 0 0), (1 1 1 1 0)}.

Considering S as a schema of a solution presentation with binary string of length m, the following is a list of basic elements in our formulation:

- $o(S)$: order of the schema S - the number of 0 and 1 (fixed) positions present in the schema. E.g., $o(S_e) = 3$.
- p_m: probability of mutation.
- $p_{sm}(S)$: probability of schema survival at mutation, $p_{sm}(S) = (1 - p_m)^{o(S)}$. Since $p_m \ll 1$, then $p_{sm}(S) \approx 1 - s(S) * p_m$.
- $\delta(S)$: defining length of the schema S – the distance between the first and the last fixed string positions. E.g., $\delta(S_e) = 4$.
- p_c: probability of crossover.
- $p_{sc}(S)$: probability of schema survival at crossover, $p_sc(S) \geq 1 - p_c * \frac{\delta(S)}{m-1}$.
- t: the time period.
- f: the fitness evaluation function, for string x_i, its fitness value is $f(x_i)$.
- $P(t)$: the population at time t, $P(t) = \{x_1, x_2, \cdots, x_{popsize}\}$.
- $f_{total}(t)$: the total fitness of the whole population at time t, $f_{total}(t) = \sum_1^{popsize} f(x_i)$.
- $f_{avg}(t)$: the average fitness of the population at the time t, $f_{avg}(t) = f_{total}(t)/popsize$.
- $f(S,t)$: the fitness of schema S at time t; assume there are p strings $\{x_{i_1}, x_{i_2}, \cdots, x_{i_p}\}$ in the population matched by a schema S at the time t, then $f(S,t) = \sum_{j=1}^{p} f(x_{i_j})/p$.
- $\xi(S,t)$: the number of strings matched by schema s at time t.

- $\xi(S, t+1) = \xi(S, t) * f(S, t)/f_{avg}(t)$.

Now, let's examine the long-term effect. Assume that the fitness of schema S remains above average by $\epsilon\%$ (i.e., $f(S,t) = f_{avg}(t) + \epsilon * f_{avg}(t)$), then $\xi(S,t) = \xi(S,0)(1+\epsilon)^t$ and we see that a schema S with an "above average" fitness level has an increasing number of strings in the next generation via the selection operation.

Combining the effects of selection, crossover, and mutation, we have the reproductive schema growth equation as follows:

$$\xi(S, t+1) \geq \xi(S,t) * f(S,t)/f_{avg}(t) * \left[1 - p_c * \frac{\delta(S)}{m-1} - o(S) * p_m\right]$$

Let us assume the population size N is large. When $N \to \infty$, the average fitness of the individuals in $P(t)$ can be described as:

$$f_{avg}(t) = \sum_{i=1}^{N} f(x^i)/N = \int_x f(x) \cdot p(x,t)dx \tag{3.1}$$

where $p(x,t)$ provides the distribution of x in the population. Similarly, the fitness of schema S at time t can be described as:

$$f(S,t) = \int_x f(x) \cdot p(x,S,t)dx \tag{3.2}$$

where $p(x,S,t)$ provides the distribution of x in schema S in the population.

Since the fitness is evaluated in the form $f(x+\Delta)$ in our proposed approach, the corresponding reproductive schema growth equation will be:

$$\xi(S, t+1) \geq \xi(S,t) * f^e(S,t)/f^e_{avg}(t) * \left[1 - p_c * \frac{\delta(S)}{m-1} - o(S) * p_m\right]$$

Assuming that x^i and Δ^i are mutually independent, $q(\Delta)$ is the continuous density function of Δ^i having defined mean value, then the effective average fitness f^e_{avg} of the population can be obtained as:

$$\begin{aligned}
f^e_{avg}(t) &= \sum_{i=1}^{N} f(x^i + \Delta^i)/N \\
&= \int_x \int_{-\infty}^{\infty} f(x+\Delta) \cdot p(x,t) \cdot q(\Delta)\, d\Delta dx \\
&= \int_x \left[\int_{-\infty}^{\infty} f(x+\Delta) \cdot q(\Delta)\, d\Delta \right] \cdot p(x,t)\, dx \\
&= \int_x f^e(x) \cdot p(x,t) dx \qquad (3.3)
\end{aligned}$$

where

$$f^e(x) \equiv \int_{-\infty}^{\infty} f(x+\Delta) \cdot q(\Delta)\, d\Delta.$$

Similarly, $f^e(S,t)$ can be described as:

$$f^e(S,t) = \int_x f^e(x) \cdot p(x,S,t) dx \qquad (3.4)$$

To put it in short, the above four equations give us the following information:

- for the original simple GA,
  - (3.1) calculates the average fitness of the whole population at the given time t,
  - (3.2) calculates the fitness of schema S at given time t,
- for the proposed GA (with perturbation),
  - (3.3) calculates the effective average fitness when consider the random noise,
  - (3.4) calculate the effective fitness of schema S when consider the random noise.

Comparing (3.1) with (3.3), and (3.2) with (3.4), we can confirm that $f^e(x)$ corresponds to $f(x)$ and we may conclude, for $N \to \infty$, that the average number of instances of each schema in the proposed GA approach increases or decreases depending on $f^e(x)$ instead of $f(x)$. We call $f^e(x)$ the effective evaluation function of $f(x)$ for the proposed GA approach. As we can see, $f^e(x)$ equals to the expected value of $f(x)$ over $x + \Delta$. Assuming that $q(\Delta)$ is symmetric, i.e. $q(\Delta) = q(-\Delta)$, then $f^e(x)$ can be rewritten as:

$$F^e(x) = \int_{-\infty}^{\infty} q(x-y) \cdot f(y)\, dy \tag{3.5}$$

Thus, the effective evaluation function for the proposed GA approach can be formulated. In practice, the population size is finite. Should the population being sufficiently large (the sufficiency depends on the distribution of the noise), the effective fitness evaluation may bear approximate characteristics as indicated in (3.5).

3.4.3. Experimental Setup for Robustness-Under-Uncertainty

Without the loss of generality, we assume a maximization problem only. The following two multi-variable functions are adapted and modified for testing the proposed robust GA approach. Function f_a (see Branke [1998] for the original function) is an n-dimensional discontinuous function with a sharp highest peak at $x_i = 1$ $(\forall\ i,\ i = 1 \cdots n)$ with a local maximum value $local\_opt_{sharp}$, and a broad lowest peak at $x_i = -1$ $(\forall\ i,\ i = 1 \cdots n)$ with a local maximum value $local\_opt_{broad}$, respectively. The local maximum value at the broad hill is designed to be 90% of the local maximum value at the sharp peak.

$(P)\quad max\ f_a(x):$

$$f_a(x) = \sum_{i=1}^{n} g(x_i),$$

$$where \qquad g(x_i) = \begin{cases} -(x_i+1)^2 + 1.17 & : -2 \le x_i < 0 \\ 1.3 * 11^{-8|x_i - 1|} & : 0 \le x_i < 2 \end{cases}$$

Function f_b (see Tsutsui and Ghosh [1997] for the original function) is an n-dimensional function with 5^n unequal peaks in the range $0 \le x_i \le 1$ $(i = 1 \cdots n)$. Function f_b is defined as follows:

$$(P) \quad max\ f_b(x):$$

$$f_b(x) = \sum_{i=1}^{n} g(x_i),$$

$$where \quad g(x_i) = \begin{cases} (e^{-2ln2(\frac{x_i-0.1}{0.8})^2} + \frac{x_i}{1.5})|\sin(5\pi x_i)|^{0.5} & : 0.4 < x_i \leq 0.6 \\ (e^{-2ln2(\frac{x_i-0.1}{0.8})^2} + \frac{x_i}{1.5})\sin^6(5\pi x_i) & : otherwise \end{cases}$$

In the following experiments, we reduce both n-variable functions to single variable functions because it is easier to observe the fitness landscape with a two-dimensional plot. We will re-iterate the simplified function definitions and show their corresponding fitness landscape plots in later sections, which detail the experiment results.

In the following eight sub-sections, we first illustrate the basics of our Robust GA to the given maximization problem: our solution representation, initialization, evaluation, genetic operators, selection process, ranking process, and parameter settings. For these simple optimization problems, we follow the guideline of Michalewicz [1995] for the solution representation and genetic operators; details are given in sections 3.4.3 - 3.4.3. The new generation creation process and ranking process are described in section 3.4.3 and 3.4.3. In section 3.4.3, we describe the common parameters. Experiment results are shown in the following section, 3.4.4.

Solution Representation

For our simple maximization problems for functions f_a & f_b, we use binary vectors as chromosomes to represent real values of the variable x. The required variable precision decides the length of the vectors. Assume that we prefer the precision to be three places after the decimal point. Taking our test function $f_{a_{1var}}$ as an example, the domain of the variable x is [-2..2]. The precision requirement implies that the domain should be divided into at least 4 * 1000 equal size ranges. This means that 12 bits are required to form our binary vector (chromosome):

$$2048 = 2^{11} < 4000 < 2^{12} = 4096.$$

It takes two steps to map a binary string $< b_{11}b_{10}\cdots b_1b_0 >$ into a real number x from the range [-2..2]:

step1: convert the binary string $< b_{11}b_{10}\cdots b_1b_0 >$ from base 2 to base 10:

$$(< b_{11}b_{10}\cdots b_1b_0 >)_2 = (\sum_{i=0}^{11} b_i * 2^i)_{10} = x'$$

step2: find a corresponding real number x:

$$x = -2 + x' * \frac{4}{2^{12}-1} ,$$

where -2 is the lower boundary of the domain and 4 is the length of the domain.

For example, the chromosome $x' = (100010110111)$ should be first converted from the base 2 to base 10:

$$x' = (100010110111)_2 = 2231;$$

and since

$$x = -2 + 2231 * \frac{4}{4095} = 0.1792,$$

we know that chromosome x' represents the number 0.1792.

Of course, the chromosomes (000000000000) and (111111111111) represent the lower bound and upper bound of the domain, -2 and 2, respectively. Since our desired precision is three places after the decimal, there is always a gap, δ, on how closely we can represent the actual target real number. For example, the closest number we can represent for the real number

$x = 1.0$ is 0.9998 (with fitness value 1.297, instead of $f_{a_{1var}}(x_{lopt_2} = 1) = 1.3$).

Initial population

Given population size (*popSize*) n, a population of n chromosomes are created, where each chromosome is a binary vector of 12 bits ($numBits = 12$). We randomly initialize all 12 bits ($prob(1) = prob(0) = 0.5$).

Evaluation function

Here, the evaluation function *eval* for binary vectors v is simply our test function. Using test function $f_{a_{1var}}$ as an example:

$$eval(v) = f_{a_{1var}}$$

where the real number x is represented by the chromosome v. As in standard genetic algorithm, the evaluation function plays the role of the environment, rating each individual solution in terms of its fitness with regard to the given problem. For example, three chromosomes with their corresponding real numbers:

$$\begin{aligned} v_1 &= (1000\ 1011\ 0111)\ , & x_1 &= 0.1792 \\ v_2 &= (1011\ 1111\ 1111)\ , & x_2 &= 1 \\ v_3 &= (0011\ 1111\ 1111)\ , & x_3 &= -1 \end{aligned}$$

Consequently, with the evaluation function $f_{a_{1var}}$, their respective fitness values are as follows:

$$\begin{aligned} eval(v_1) &= f_{a_{1var}}(x_1) &&= 0.0137 \\ eval(v_2) &= f_{a_{1var}}(x_2) &&= 1.3 \\ eval(v_3) &= f_{a_{1var}}(x_3) &&= 1 \end{aligned}$$

If our objective is to find the maximum value for the given problem, it is clear that $v_2(x_2)$ is the best chromosome (solution) of the three, since it has the highest value in terms of the evaluation function $f_{a_{1var}}$. Again, due to the three-decimal-place precision, there is a gap, $\delta = 0.0002$, between our representation and the actual target real number. Hence the best solution in the experiment could only be $f_{a_{1var}}(0.9998) = 1.29675$, instead of $f_{a_{1var}}(1.0) = 1.3$.

Genetic Operators

Two classical genetic operators, mutation and crossover, are used during the alteration phase of our genetic algorithm:

- mutation: with a probability equal to the mutation rate, altering one or more genes (bits) in a chromosome. The mutation rate is the probability of mutation for each single bit in the chromosome. Therefore, given a mutation rate 0.01, chromosome length 20, population size 20, we expected 4 bits to mutate in each generation. Assume that a mutation happens on the ninth gene of the previously mentioned chromosome v_1. Since the ninth bit in v_1 is a 0, it would be flipped to 1. Thus the chromosome v_1 after the mutation would be:

  $$v_1' = (1000\ 1011\ 1111)$$

  The mutated chromosome represents the value x_1' = -0.8132 and its objective function value $f_{a_{1var}}(x_1') = 0.9651$. This means that this particular mutation resulted in a significant increase of the fitness value of the original chromosome, v_1.

  On the other hand, if the third gene of v_1 is selected for a mutation, since the fifth bit in v_1 is 1, it would be flipped into 0. Thus the chromosome v_1 after the mutation would be:

  $$v_1'' = (1000\ 0011\ 0111)$$

The mutated chromosome represent the value $x_1'' = 0.0620$ and its objective value $f_{a_{1var}}(x_1'') = 0.0072$, a decrease of the fitness value that was originally $f_{a_{1var}}(x_1) = 0.1792$.

- crossover: two parent chromosomes with a selected exchange point (for instance, after the fourth gene), each parent keeps the first segment (from bit 1 up to the bit right before the exchange point, or bit 4 in our example) of its own chromosome, and concatenate with the partner's second segment (the bit right after the exchange point up to the end of the chromosome, which here would be bit 5 to bit 12). Let us illustrate the crossover operator on the two boundary chromosomes, assuming that the crossover point was randomly selected after the second gene:

  v_{lb} = (00 | 00 0000 0000),

  v_{ub} = (11 | 11 1111 1111).

  The two resulting offsprings are:

  v'_{lb} = (0011 1111 1111) and

  v'_{ub} =(1100 0000 0000).

  These offsprings evaluate to:

  $eval(v'_{lb}) = f_{a_{1var}}(-1.0007) = 1.0$ and

  $eval(v'_{ub}) = f_{a_{1var}}(+1.0007) = 1.2947$.

  Comparing to the parents which evaluate to:

  $eval(v_{lb}) = f_{a_{1var}}(2) = 0.0051$ and

  $eval(v_{ub}) = f_{a_{1var}}(-2) = 0$.

  Both offsprings have better evaluations than their parents.

Creation of New Generation

Three more operators, *elitism*, *selection* and *elimination*, are used together with *mutation* and *crossover* during the alteration phase of our genetic algorithm:

- elitism: for the existing population, the top four solutions are selected and added into the next generation population first.

- selection: using tournament-2 selection, two parent solutions are selected as mating candidates. Tournament-2 selection is implemented by randomly selecting two individuals from the current population. The individual with the higher fitness value is then considered the winner and used for the next process (mating). For generating one pair of candidate solutions, we need to perform the tournament-2 selection twice.
- mating: for a pair of candidate solutions, each solution goes through mutation individually first (with the given mutation probability p_m). The candidates then undergo crossover with a given crossover probability, p_x.
- elimination: The two parent candidates may or may not actually mutate or undergo crossover. In the case that no actual mating takes place, the mutated parents (which in some cases may not be mutated) are kept in a temporary pool. Should the crossover event actually happen, the two generated children solutions are kept in the temporary pool. Together with the two original parent candidates, there are four solutions in the temporary pool. The two solutions with lower fitness values are eliminated.
- building the new generation: the survivors of the elimination process (the two solutions with higher fitness values) are added into the next generation population.

Elitism is done once to secure the spots in the next generation for the best four solutions known in current population. The selection, mating, elimination, and building the new generation processes are repeated until we collect $popSize$, n, solutions for the new generation.

Solution Sampling and Collection

We simulate the noisy environment by perturbing the decision variable. Let us define "solving one perturbed problem" as solving a problem with one version of perturbed decision variable. With the knowledge about uncertainty of a given problem's decision variable, we generate a set of sample noise Δs, $DSample$. For each member $\Delta_i \in DSample$, we solve the problem with a perturbed decision variable to its optimality. Since the intuition of our approach is "searching for the most popular solution", we record the $NtopSoln$ best solutions for each perturbation with its corresponding log. After all $Npert$ perturbed problems are solved, a final tally is taken for all solutions that have ever shown up in any of the individual N_{pert} logs. The solutions appearing most frequently are considered the "most popular" solutions. Using the number of perturbed cases in which a particular solution appears as the solution's performance measure, a "more popular" solution will score higher in our ranking system. Should two solutions have the same appearing frequency (for instance, solution x_1 and solution x_2 both appear in 68 perturbed cases), the decision-maker can always consider the other information such as their fitness values with regard to the objective function in the non-perturbed case.

Parameters

The following is a list of common GA parameters used for our test function $f_{a_{1var}}$ and $f_{b_{1var}}$:

- pop_size: population size,
- len_{soln}: solution length - the number of bits in the binary string solution representation. The variable value precision is decided by the solution length.
- p_m: probability of mutation,
- p_x: probability of crossover,
- g: number of generations,
- t: number of trials, each trial starts with one randomly initialized population, and evolves through g generations,
- L: rejection level - the minimum acceptable sub-optimality of solutions (e.g., $L = 0.75$ indicates rejecting the lower 75% solutions, *viz.*, accepting only the solutions with third quartile fitness values).

- $N_{topSoln}$: number of recorded top solutions for each perturbed problem, $N_{topSoln}$ = (1-L)*size(solution space).

- N_{pert}: number of perturbed problems,

- σ: the standard deviation of the normally distributed error caused by the environmental noise. For example, with x range [-2, 2], $sigma = 0.01$ equivalent to 0.25% error in x domain.

3.4.4. Experiment Results for Robustness-Under-Uncertainty

Having shifted our view point to the effect of environmental noise on the genotype—phenotype mapping, we now focus on the possible error of our decision variable in the objective function $f(x)$. The expected error δ on the decision variable causes the function shift in the horizontal direction, presented by $f'(x)$ (i.e. $f'(x) = f(x + \delta)$). Let us assume that the estimated error, δ, is normally distributed with mean zero and standard deviation σ, i.e. $\delta \sim \mathcal{N}(0, \sigma^2)$. The relationship of $f(x)$ and $f'(x)$ is shown in Figure 16.

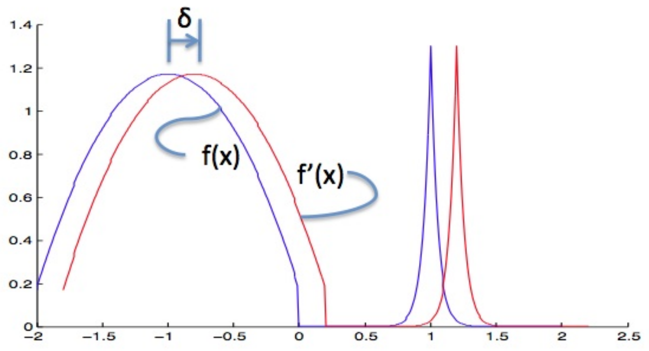
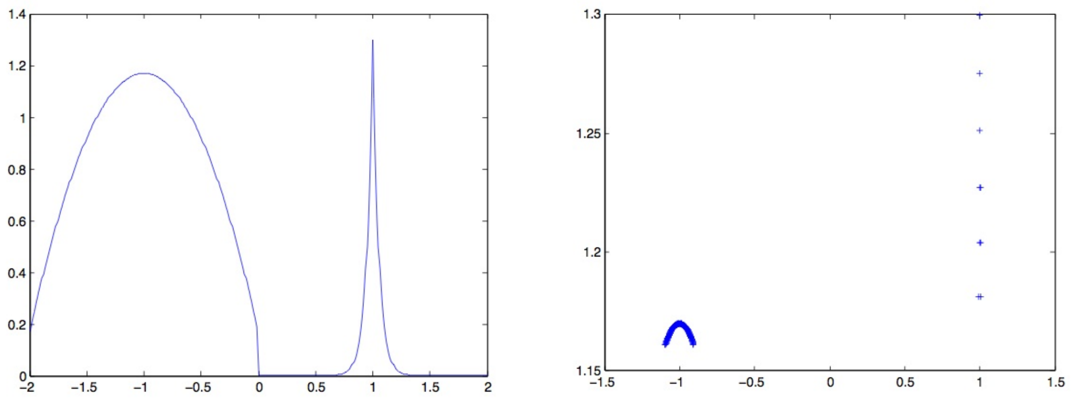


Figure 16: Function $f_{a_{1var}}$ with and without decision variable perturbation



(a) Function $f_{a_{1var}}$ (b) Top 5% solutions of function $f_{a_{1var}}$

Figure 17: Function $f_{a_{1var}}$ Fitness Values vs X values

Test Function $f_{a_{1var}}$

The simplified function $f_{a_{1var}}$ is defined as:

$$f_{a_{1var}}(x) = \begin{cases} -(x+1)^2 + 1.17 & : -2 \leq x < 0 \\ 1.3 * 11^{-8|x-1|} & : 0 \leq x < 2 \end{cases}$$

The fitness values of the solutions to function $f_{a_{1var}}$ are plotted in Figure 17(a). The top 5% solutions' x values and their fitness values are shown in Figure 17(b).

Given the parameter settings as shown in Table 28, assume that we are interested in all solutions with top 1.25% fitness values ($L = 0.9875$), which equals to the top 50 solutions since $f_{a_{1var}}$'s domain size is 4000 with three-decimal-place precision. We observe the solutions recommended by our robust GA approach in the following four scenarios: (1) with no uncertainty of the knowledge about our decision variable, (2) with a possible 0.25% domain range error ($\sigma = 0.01$), (3) with a possible 1.25% domain range error ($\sigma = 0.05$), and (4) with a possible 2.5% domain range error ($\sigma = 0.1$).

$f_{a_{1var}}$ Experiment Results:

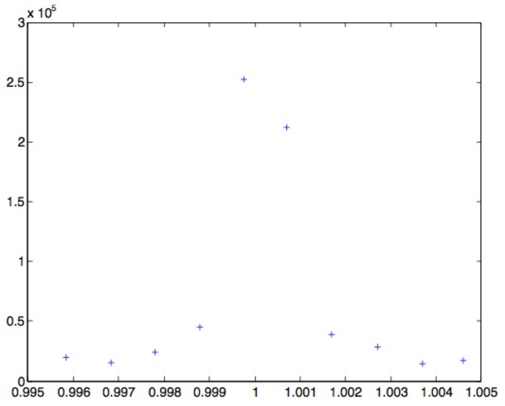
Fixed Value Parameters		Non-Fixed Value Parameters
parameter	value	parameter
pop_size	20	
len_{soln}	10	σ:
p_m	0.15	0
p_x	0.3	0.01
g	150	0.05
t	10	0.1
N_{pert}	100	
L ($N_{topSoln}$)	0.9875(50)	

Table 28: Parameter settings for $f_{a_{1var}}$ experiments

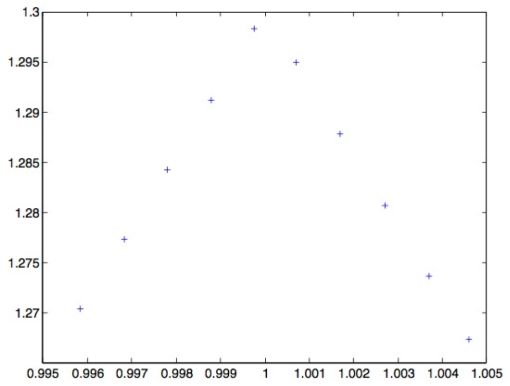
1. Zero Uncertainty: with the simple GA, the best solutions are found all around $x = 1$ with disproportionate high frequency as expected, see Figure 18(a). The objective values ranging from the global maximum 1.2983 (the maximum objective value one can actually get for the given problem with required x value precision to three decimal places) to 1.2673 (at $x = 1.0046$): see Figure 18(b). With our proposed approach, the top 46 recommended solutions all have robustness score equal to 100, including ten solutions around$x = 1$ and the rest around $x = -1$. As shown in Figure 18(c), the top 50 recommended solutions are considered more or less equally good. The fitness values of the top 50 recommended solutions are shown in Figure 18(d).

2. With a 0.25% error: given that the estimated standard deviation is 0.01 (0.25% of the x domain size), our robust GA approach highly recommends the solutions around $x = -1$. Figure 19(a) shows the recommended solutions with the corresponding robustness-under-uncertainty scores, R_U, for the case of perturbation with $\sigma = 0.01$. Figure 19(b) shows that all top 30 recommended solutions gathered around $x = -1$ neighborhood.

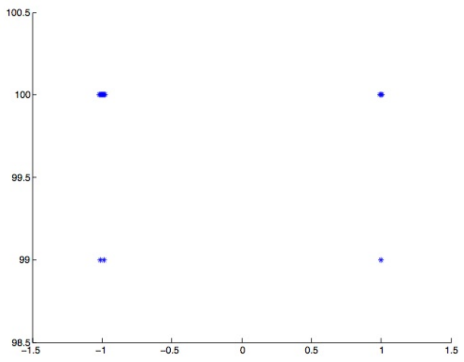
3. With an 1.25% error: given that the estimated error standard deviation is 0.05 (1.25% of the x domain size), the recommended solutions further cluster around $x = -1$. Figure 19(c) shows the recommended solutions with their robustness scores for the case of perturbation with $\sigma = 0.05$. Figure 19(d) shows that all top 100 recommended



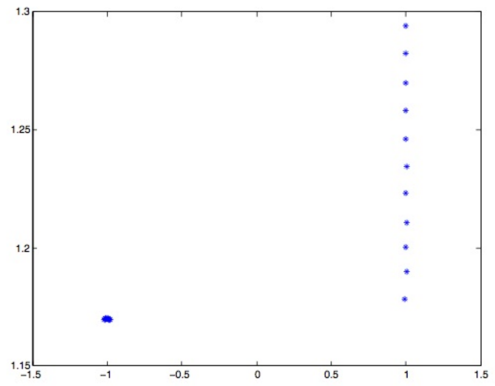
(a) Top 10 Solns Sampling Frequency



(b) Top 10 Solns Fitness Values



(c) Top 50 Solns R_U Scores



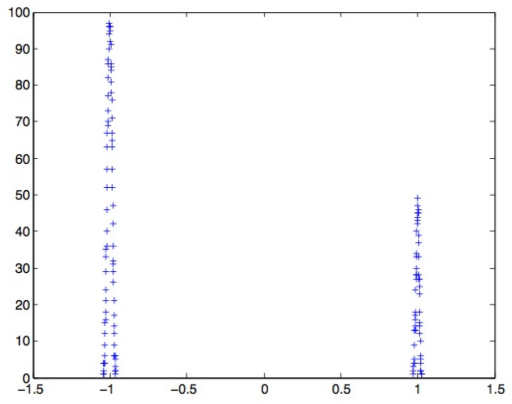
(d) Top 50 Solns Fitness Values

Figure 18: $f_{a_{1var}}$ with Zero Error, Top Solutions' R_U Scores and Fitness Values

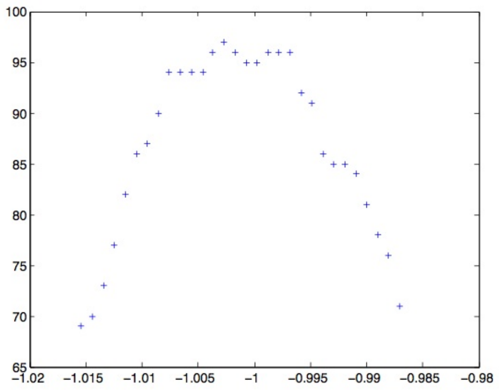
solutions gathered around $x = -1$ neighborhood. Further examining the collected solution, we find that all top 147 recommended solutions belong to the $x = -1$ neighborhood.

4. With a 2.5% error: given that the estimated error standard deviation is 0.1 (2.5% of the x domain size), the recommended solutions concentrate around the $x = -1$ area as expected. Figure 19(e) shows the recommended solutions with their robustness scores for the case of perturbation with $\sigma = 0.1$. Figure 19(f) shows that all top 150 recommended solutions gather around the $x = -1$ neighborhood. Further examination reveals that all top 194 recommended solutions belong to the $x = -1$ neighborhood.

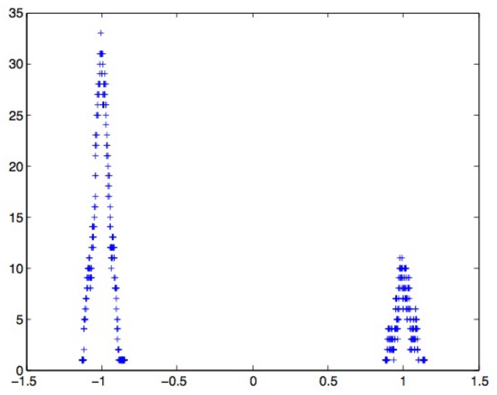
Table 29 shows the search outcome with both x variable precision set at three and four decimal places. Column 1 indicates the estimated error size of x value, and Column 2 shows that around the $x = -1$ neighborhood, the best found solution's rank and R_U score. The rank of a given solution x indicates how many other solutions have higher fitness values than its own. The R_U score of a given solution x is the number of perturbations in which the solutions appears in the corresponding perturbation's SoIs set. Column 3 shows similar information for solutions around the $x = 1$ neighborhood. Column 4 is the ratio $\frac{R_{U_{Peak-1}}}{R_{U_{Peak+1}}}$. Since the proposed approach is based on the appearance frequency, we believe the ratio between R_Us of the top solutions in different neighborhood is more meaningful than the actual score value. Table 29 shows that the experiment results for the x variable with precision of four decimal places are in line with the results for setting the x variable precision to three decimal places. As the estimated error of x increases, more and more top recommended solutions gather around the broader peak at $x = -1$, while solutions around the sharp peak ($x = 1$) are pushed down on the recommendation list gradually. The emphasis of the top solutions around the broader peak ($x = -1$) can be seen as the R_U ratio increasing as the error size increases.



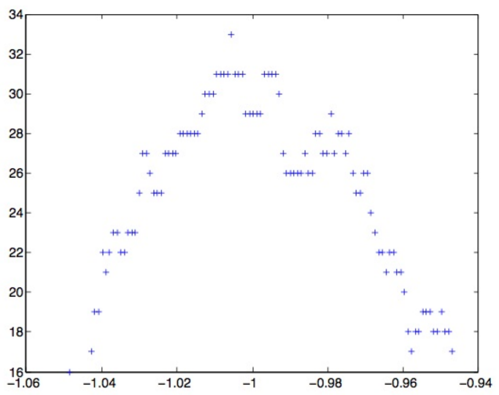
(a) σ=0.01, Top 100 Solns R_U Score



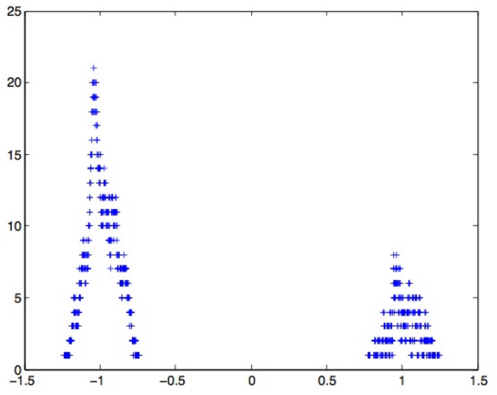
(b) σ=0.01, Top 30 Solns Fitness Value



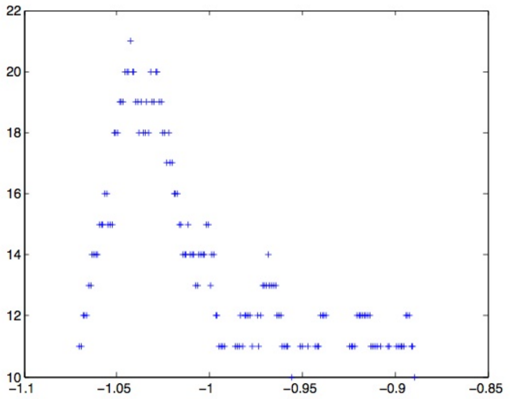
(c) σ=0.05, Top 100 Solns Robustness Score



(d) σ=0.05, Top 100 Solns Fitness Value



(e) σ=0.1, Top 150 Solns Robustness Score



(f) σ=0.1, Top 150 Solns Fitness Value

Figure 19: $f_{a_{1var}}$ with different size x error, Top Solutions' R_U Scores and Fitness Values

σ value	Ranks (Rk) & Scores (R_U) of Top Solutions in the Neighborhood of:		R_U Ratio
	Peak -1	Peak +1	
x variable precision: three decimal places			
0	$Rk: 1, R_U$=100	$Rk: 1, R_U$=100	1
0.01	Rk: 1, R_U=97	Rk: 40 , R_U=49	1.9796
0.05	Rk: 1, R_U=33	Rk: 123, R_U=15	2.2
0.1	Rk: 1, R_U=21	Rk: 195, R_U=8	2.6250
x variable precision: four decimal places			
0	$Rk: 1, R_U$=100	$Rk: 1, R_U$=100	1
0.001	Rk: 1, R_U=100	Rk: 1 , R_U=100	1
0.005	Rk: 1, R_U=100	Rk: 128, R_U=73	1.3699
0.01	Rk: 1, R_U=92	Rk: 177, R_U=41	2.2439
0.05	Rk: 1, R_U=34	Rk: 495, R_U=13	2.6154
0.1	Rk: 1, R_U=24	Rk: 752, R_U=9	2.6667

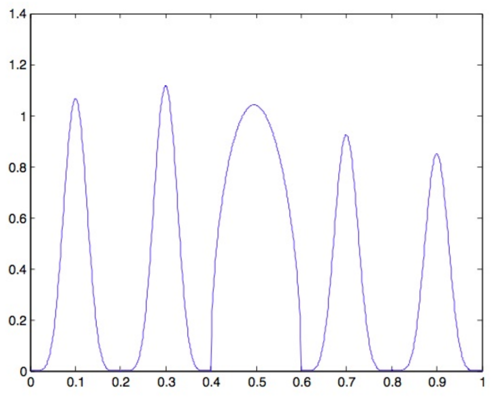
Table 29: $f_{a_{1var}}$, x value with 3 and 4 decimal place precision, search outcome with various x value error sizes

Test Function $f_{b_{1var}}$

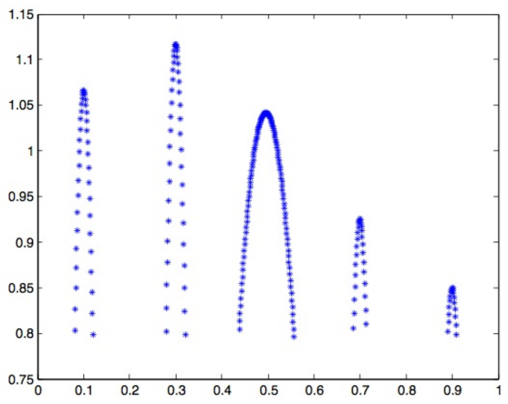
The simplified function $f_{b_{1var}}$ is defined as follows:

$$f_{b_{1var}}(x) = \begin{cases} (e^{-2ln2(\frac{x-0.1}{0.8})^2} + \frac{x}{1.5})|\sin(5\pi x)|^{0.5} & : 0.4 < x \leq 0.6 \\ (e^{-2ln2(\frac{x-0.1}{0.8})^2} + \frac{x}{1.5})\sin^6(5\pi x) & : otherwise \end{cases}$$

Given that the variable x can take values from a domain [0,1], the simplified function $f_{b_{1var}}$ is a function with five peaks locating at (around) $x = 0.1, 0.3, 0.5, 0.7, 0.9$, respectively. The fitness values of the solutions to function $f_{b_{1var}}$ is plotted in Figure 20 (a). Assuming that three-decimal-place precision is desired, by ranking the solutions with their corresponding fitness values we know that the top 250 solutions cover all five local optima in $f_{b_{1var}}$. Suppose precision with either three or four decimal places for the variables' value is desired. With binary string solution representation, we list the local optima, ranking of each local optimum, and their corresponding x values for function $f_{b_{1var}}$ in Table 30. Again, the ranking of a given solution x indicates how many other solutions have higher R_U score than its own. This information is useful for monitoring the performance of our proposed



(a) Function $f_{b_{1var}}$



(b) $f_{b_{1var}}$ Top 250 Solutions

Figure 20: Function $f_{b_{1var}}$ Fitness Values vs x Values

approach.

3 Decimal Place Precision				4 Decimal Place Precision			
local opt.	x value	fit. val.	rank	neighborhood	x value	fit. val.	rank
$lopt_1$	0.1	1.0667	16	$peak01$	0.1002 - 0.1006	1.0668	158
$lopt_2$	0.3	1.1170	1	$peak03$	0.2997 - 0.3001	1.1170	1
$lopt_3$	0.5	1.0404	32	$peak05$	0.4945 - 0.4968	1.0419	308
$lopt_4$	0.7	0.9252	144	$peak07$	0.6996	0.92572	1432
$lopt_5$	0.9	0.8500	202	$peak09$	0.8998, 0.8999	0.85001	2009

Table 30: $f_{b_{1var}}$ local optima x values vs. fitness values

To understand how some factors influence our search outcome, we consider the following three parameters:

1. Rejection Level L: the unwillingness to settle on solution that has fitness value lower than a certain level, e.g., with L= 0.95, one accepts only solutions with top 5% fitness values in the candidate pool and rejects the remaining 95% of solutions. Lower L value indicates acceptance of solution with lower fitness value in exchange for higher robustness, i.e., with L=0.9, one accepts the top 10% of solutions, while with L=0.75, one is willing to consider all the solutions with top 25% of fitness values in the alternative solution pool.

2. Estimated Error Size σ: assuming the estimated error is normally distributed with mean at 0 and unknown standard deviation σ. One could set the estimated error size proportional to the size of the decision variable domain.

3. Solution Length len_{soln}: the number of bits for the binary string solution representation. len_{soln} decides the precision of variable value. Meanwhile, the precision of variable value influences the number of potential solutions in a fixed range of variable domain.

First, we look at the effect of different rejection levels L. Given three-decimal-place precision for x's value, the experiment parameter settings are listed in Table 31.

Fixed Value Parameters		Non-Fixed Value Parameters
parameter	value	parameter
pop_size	20	L ($N_{topSoln}$):
p_m	0.15	0.95(50)
p_x	0.3	0.90(100)
g	150	0.85(150)
t	10	0.80 (200)
N_{pert}	100	0.75 (250)
σ	0.01	
len_{soln}	10	

Table 31: Parameter settings for $f_{b_{1var}}$ rejection level experiments

Assuming the estimated error size is 0.05, 5% of the x domain ([0,1]). By varying the rejection level from 0.95 to 0.70, we increase the number of SoIs, which we keep tracking throughout the one hundred perturbations. Table 32 lists the experiment results with different rejection levels. Details about each column are as follows:

- column 1, L: the rejection level, which ranges between [0.95, 0.75]. The rejection level determines the number of solutions of interest (SoIs) that we keep track throughout all perturbations. A high rejection level means tracking fewer solutions of a perturbation during the process (i.e., L=0.95 indicates tracking only the solutions with top 5% fitness values in each perturbation's SoIs set).

- column 2 to 6, $Peak0X$: shows the rank (Rk) and score (R_U) of the best solution around the three peaks at $x = 0.5$, 0.3, 0.1, 0.7, and 0.9.
- column 7, N: the number of SoIs that are tracked through each perturbation. N is decided by L and variable value precision (len_{soln}).

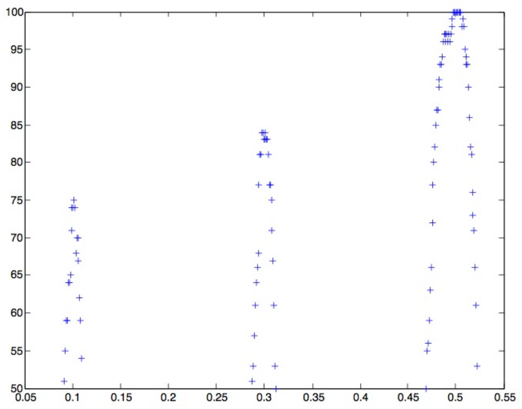
L	Rank Distribution of Top Ranked Solutions in the Neighborhood of:					N
	Peak 05	Peak 03	Peak 01	Peak 07	Peak 09	
.95	Rk:14,R_U=57	Rk: 1,R_U=66	Rk:36,R_U=46	Not Found	Not Found	50
.90	Rk: 1,R_U=100	Rk:38,R_U=79	Rk:61,R_U=69	Not Found	Not Found	100
.85	Rk: 1,R_U=100	Rk:55,R_U=93	Rk:73,R_U=86	Rk:251,U_R=8	Not Found	150
.80	Rk: 1,R_U=100	Rk:69,R_U=93	Rk:77,R_U=91	Rk:226,R_U=35	Rk:351,R_U=5	200
.75	Rk: 1,R_U=100	Rk:64,R_U=96	Rk:90,R_U=93	Rk:138,R_U=81	Rk:194,R_U=66	250

Table 32: $f_{b_{1var}}$, recommended solution ranks and scores with different rejection level value.

The effect of changing rejection level: At the 0.95 rejection level, only solutions with the top 5% fitness values of each perturbation are tracked through the process. As a result, solutions around (and including) local optimum $lopt_3$ (at x = 0.5) have a lower chance of being in the tracked SoIs group. The unwillingness to settle for lower fitness values reflects the outcome that recommends solutions around peak03, slightly over the solutions around peak05. Table 32 row L=0.95 shows that the top 13 recommended solutions are around Peak03, with top R_U=66, followed by solutions around Peak05, with top R_U=57.

After lowering the rejection level to 0.90, the top 37 recommended solutions are all located around the peak $x = 0.5$ (Figure 21). Further lowering the rejection level did not change the pattern that solutions around peak $x = 0.5$ are always in the top recommended solution group, while solutions around other peaks are gradually pushed down the recommendation list. Figure 22 shows that there are fewer solutions around Peak03 and Peak01 in the top 100 recommended solutions when one is willing to accept lower fitness value in exchange for solution robustness. The pattern of recommended solutions shifting toward the broader peak as estimated error increases is in line with our results of test problem $f_{a_{1var}}$.

More importantly, the results in Table 32 show that the proposed approach finds solutions around all local optima as long as the local optimum's fitness value is acceptable to the


(a) Solutions X Values vs R_U Scores

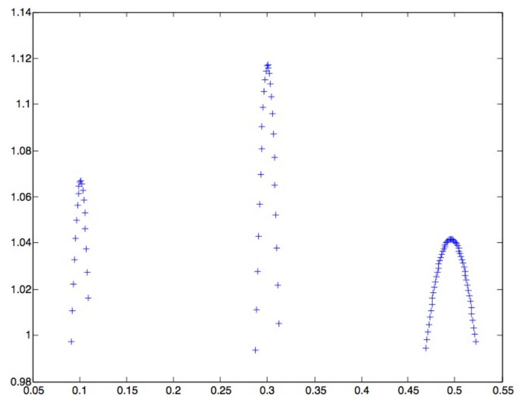
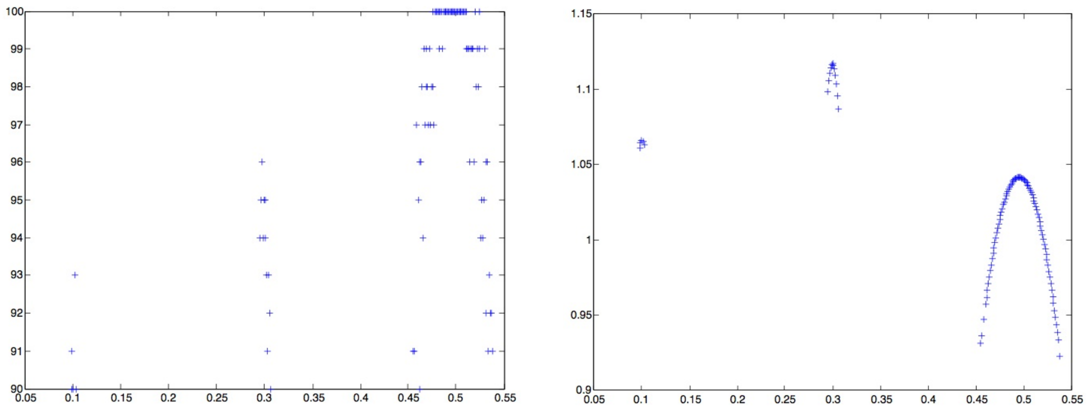

(b) Solutions X Values vs Fitness Values

Figure 21: $f_{b_{1var}}$ L = 0.90 σ = 0.01 Top 100 Solutions

decision-maker. For example, Table 30 (row 6, column 4) indicates that $lopt_4$'s ranking is 144. Naturally, given the L value set on either 0.95 or 0.9, one considers only the sampled solutions with fitness values among either the top 50 or the top 100. Therefore, the suggested approach recommended no solution around Peak07 and Peak09, the two peaks with low-fitness-value local optima. On the other hand, when the rejection level is lowered to 0.85 or 0.80, our approach does find solutions around Peak07 and Peak09, correspondingly. To verify that such findings are not random luck, we repeat the experiment with x variable precision set at four decimal places; the results are in line with our experiment of lower x variable precision. Table 30 (row six and row seven, column eight) indicates that the best solution around Peak07 is ranked 1432, while the best solution around Peak09 is ranked 2009. Four-decimal precision of an x variable value produces a total of ten thousand candidate solutions. Even with zero perturbation, the proposed approach finds solutions around Peak07 with rejection level set at 0.9 (one tracks only the top 1000 sampled solutions). Our approach also finds solutions around Peak09 with rejection level set at 0.85. This finding confirms the efficiency of the suggested Robust GA on finding solutions around all local optima, which have fitness values within the acceptance level.

Next, we look at the effect of different estimated error sizes and variable precisions. Keeping



(a) Solutions X Values vs R_U Scores

(b) Solutions X Values vs Fitness Values

Figure 22: $f_{b_{1var}}$ L = 0.75 σ = 0.01 Top 100 Solutions

all GA related parameters unchanged, we set the rejection level at 0.90 (accepting only the top 10% solutions). Experiment parameter settings are listed in Table 33. Since increasing the variable precision also means extending the search space to a certain extent, we examine the effect of error size on the search outcome with variable precision of three, four, and five decimal places. For each peak at different x values (i.e., x = 0.5, 0.3, 0.1,..., etc.), we record the R_U value of the best solution found around it respectively. Table 34 shows the test results with variable precision of three decimal places. For complete test results, see Appendix Table 41. In Table 34, column 2, 3, and 4 record the best solution's rank and R_U score found around peaks at x = 0.5, 0.3, and 0.1. We find that, with three-decimal-place precision for variable x, given error size $\sigma = 0.001$, our approach ranks 62 solutions around Peak05, Peak03, and Peak01 equally well, all with R_U=100. As the error size increased, the solutions around Peak05 start to win over the ones around other peaks. Given error size 0.0075 (Table 34 row 7 column 3), we see the best solution is still found around Peak05, while the second best solution is found around Peak3: it is ranked 33^{th} with R_U=91 (rank 33 indicates that there are 32 solutions around Peak05 that are considered better than it).

While there are a total of five peaks in the x domain [0,1] for test function $f_{b_{1var}}$, our proposed GA performs as expected in finding high quality solutions with regard to both

Fixed Value Parameters		Non-Fixed Value Parameters
parameter	value	parameter
pop_size	20	σ:
p_m	0.15	0.001
p_x	0.3	0.005
g	150	0.01
t	10	0.025
N_{pert}	100	0.05
L	0.9	0.075
len_{soln}	10, 14, 17	0.1

Table 33: Parameter settings for $f_{b_{1var}}$ error size experiments

	Rank Distribution of Top Ranked Solutions in the Neighborhood of:			Ratio of Top 100 Solutions
σ value	Peak 0.5	Peak 0.3	Peak 0.1	P05:P03:P01
0.001	28 solutions R_U=100	21 solutions R_U=100	13 solutions R_U=100	55 : 25 : 20
0.0025	34 solutions R_U=100	13 solutions R_U=100	7 solutions R_U=100	55 : 25 : 20
0.005	Rk: 1, R_U=100	Rk: 28, R_U=99	Rk: 43 R_U=97	54 : 26 : 20
0.0075	Rk: 1, R_U=100	Rk: 33, R_U=91	Rk: 51 R_U=81	55 : 25 : 20
0.01	Rk: 1, R_U=100	Rk: 35, R_U=84	Rk: 55 R_U=71	55 : 27 : 18
0.025	Rk: 1, R_U=78	Rk: 62, R_U=45	Rk: 98, R_U=35	74 : 25 : 1
0.05	Rk: 1, R_U=45	Rk: 83, R_U=28	Rk: 128, R_U=25	100 : 0 : 0
0.075	Rk: 1, R_U=40	Rk: 58, R_U=27	Not in top 100 solns	78 : 22 : 0
0.1	Rk: 1, R_U=30	Rk: 32, R_U=25	Not in top 100 solns	70 : 30 : 0

Table 34: $f_{b_{1var}}$, error size experiment with three-decimal-place variable precision

fitness value and solution robustness. The test results show that all top recommended solutions are around Peak05 (high robustness) and Peak03 (high fitness value). Therefore, we focus on the recommended solutions around Peak05 and Peak03. Again, since the R_U score is a "popular vote count", the ratio of R_Us of top solutions around different peaks is more meaningful than the face value of R_U itself. Table 35 shows the relevant information extracted from the test results.

	3 Decimal Place Precision			4 Decimal Place Precision			5 Decimal Place Precision		
σ	$R_{U_{P05}}$	$R_{U_{P03}}$	$R_{R_{U_{P05P03}}}$	$R_{U_{P05}}$	$R_{U_{P03}}$	$R_{R_{U_{P05P03}}}$	$R_{U_{P05}}$	$R_{U_{P03}}$	$R_{R_{U_{P05P03}}}$
0.001	100	100	1.0000	99	100	0.9900	85	85	1.0000
0.0025	100	100	1.0000	95	98	0.9694	81	79	1.0253
0.005	100	99	1.0101	98	94	1.0426	78	68	1.1471
0.0075	100	91	1.0989	94	81	1.1605	76	61	1.2459
0.01	100	84	1.1905	92	70	1.3143	76	47	1.6170
0.025	78	45	1.7333	63	38	1.6579	58	32	1.8125
0.05	45	28	1.6071	43	24	1.7917	34	20	1.7000
0.075	40	27	1.4815	31	19	1.6316	33	19	1.7368
0.1	30	25	1.2000	26	26	1.0000	23	17	1.3529

Table 35: $f_{b_{1var}}$, $R_{U_{P05}}$:$R_{U_{P03}}$ Ratio with Different Variable Precision

The effect of changing error size: we find that with small error size (σ <0.0025, 0.25% of x domain), solutions around the peak at $x = 0.3$ (Peak03) slightly win over solutions around the peak at $x = 0.5$ (Peak05). It indicates that small shifting on the very top of the peak does not affect the fitness value much, so the solutions around sharper peak at $x = 0.3$ are equally recommended as the solutions around the broader peak at $x = 0.5$. As the error size increased, the robustness of solutions around Peak05 weights more, and the ratio of Peak05 top solution's R_U to Peak03 top solution's R_U increases from around 1 to over 1.5. For example, in Table 35 column 7, the $R_{R_{U_{P05P03}}}$ value increases as the error size rises from 0.001 to 0.025. Notably, $R_{R_{U_{P05P03}}}$ value decreases as error size increased over 0.05 (σ >0.05, 5% of x domain). Because the test problem $f_{b_{1var}}$ has a half-peak width of 0.1, $\sigma > 0.05$ indicates a greater chance that the shifting of x value will cause the solution to move into its neighboring peaks. As a result of over-shifting variable x, the R_U ratio between top solutions around Peak05 and Peak03 decreases, and the recommendation is less definitive.

The effect of increasing variable precision: we examine the ratio of top solutions' R_Us at Peak05 and Peak03. Table 35 lists the R_Us of the best solutions around Peak05 and Peak03 with different error size (σ), grouped by the variable value precision. Column 1 specifies the error size σ, ranging from 0.001 to 0.05. Column 2, 3, and 4 are the results of an experiment with x variable of three-decimal-place precision. Column 2 records a R_U score of the best solution found around Peak05, while Column 3 records R_U score of the best solution found around Peak03. Column 4 shows the ratio, $R_{R_{U_{P05P03}}}$, of the top solutions' R_U scores, where $R_{R_{U_{P05P03}}} = \frac{R_{U_{Peak05}}}{R_{U_{Peak03}}}$. Column 5, 6, and 7, and Column 8, 9, and 10 present similar information for experiments with x variable of four- and five-decimal-place precision.

As shown in Table 35, for a given error size, $R_{R_{U_{P05P03}}}$ seems to follow the trend of increasing its value as the precision decimal place increases. Therefore, our null hypothesis of interest here is that of zero shift (θ) in $R_{R_{U_{P05P03}}}$ value due to the increasing variable precision:

$$H_0: \ \theta = 0$$

versus

$$H_1: \ \theta > 0$$

Using Wilcoxon one-sided upper-tail test, we examine the following three comparisons:

- Increasing variable precision from 3 decimal places to 4 decimal places.
- Increasing variable precision from 4 decimal places to 5 decimal places.
- Increasing variable precision from 3 decimal places to 5 decimal places.

We find that neither increasing variable precision from three to four decimal places nor increasing variable precision from four to five decimal places leads to any statistically significant shift in $R_{R_{U_{P05P03}}}$ values (p-value is 0.234 and 0.148 respectively). But we do find significant increase in $R_{R_{U_{P05P03}}}$ values in the case of increasing precision from three to five decimal places (p-value = 0.016). The increasing of $R_{R_{U_{P05P03}}}$ value indicates the solution robustness (Peak05) is weighted more than the solution fitness value (Peak03) as the variable precision increases.

3.4.5. Multi-Variable Optimization Problem

In this section we extend our single-variable optimization problems to two-variable cases and examine the performance of our robust GA.

Function $f_{a_{2var}}$

The function $f_{a_{2var}}$ is defined as:

$$f_{2var}(x) = \sum_{i=1}^{2} g(x_i),$$

$$where \qquad g(x_i) = \begin{cases} -(x_i+1)^2 + 1.235 & : -2 \le x_i < 0 \\ 1.3 * 11^{-8|x_i - 1|} & : 0 \le x_i < 2 \end{cases}$$

Experiment parameter settings are listed in Table 36

Fixed Value Parameters		Non-Fixed Value Parameters
parameter	value	parameter
pop_size	100	σ:
p_m	0.07	0
p_x	0.25	0.01
g	400	0.05
t	10	0.1
N_{pert}	100	
L	0.9975	
len_{soln}	12	

Table 36: Parameter settings for $f_{a_{1var}}$ error size experiments

By tracking only the top 0.25% of sampled solution, the proposed robust GA performs similarly as it does with single-variable problems. We see greater preference for solutions around the broader peak as the top recommended solutions shifted from Peak(+1,+1) to

Peak(-1, -1) while the estimated error size increases. The search results are shown in Table 37.

σ value	Ranks (Rk) & Scores (R_U) of Top Solutions in the Neighborhood of: Peak(-1, -1)	Peak (+1, +1)	R_U Ratio
x variable precision: three decimal places			
0	$Rk: 857, R_U$=15	$Rk: 10, R_U$=67	0.2239
0.01	Rk: 1, R_U=20	Rk: 15 , R_U=14	1.4286
0.05	Rk: 1, R_U=12	Rk: 112, R_U=5	2.4
0.1	Rk: 1, R_U=7	Rk: 10, R_U=5	1.4

Table 37: $f_{a_{2var}}$, x value with 3 decimal place precision, search outcome with various x value error sizes

Function $f_{b_{2var}}$

The function $f_{b_{2var}}$ is defined as:

$$f_{b_{2var}}(x) = \sum_{i=1}^{2} g(x_i),$$

$$where \qquad g(x_i) = \begin{cases} (e^{-2ln2(\frac{x_i-0.1}{0.8})^2} + \frac{x_i}{1.5})|\sin(5\pi x_i)|^{0.5} & : 0.4 < x_i \leq 0.6 \\ (e^{-2ln2(\frac{x_i-0.1}{0.8})^2} + \frac{x_i}{1.5})\sin^6(5\pi x_i) & : otherwise \end{cases}$$

Experiment parameter settings are listed in Table 36

Fixed Value Parameters		Non-Fixed Value Parameters
parameter	value	parameter
pop_size	100	σ:
p_m	0.07	0
p_x	0.25	0.01
g	400	0.025
t	10	0.05
N_{pert}	100	
L	0.999	
len_{soln}	12	

Table 38: Parameter settings for $f_{b_{2var}}$ error size experiments

The experiment results of two-variable test problem $f_{b_{1var}}$ are shown in Table 39. The performance of the proposed robust GA is in line with the results of solving the single-variable problem $f_{b_{1var}}$. We find that small size (or zero) error results in a higher amount of recommended solutions from the Peak33 ($x_1 = 0.3$, $x_2 = 0.3$) neighborhood. As the error size increases, more recommended solutions are from the Peak55 ($x_1 = 0.5$, $x_2 = 0.5$) neighborhood, while solutions around Peak33 are less favored.

However, when the error size is relatively large (at 0.05, which is 50% of the peak half width), we see the similar effect as in the single variable test problem $f_{b_{1var}}$, where the recommendation validity decreases as the error size is large enough to push the x variable from a certain peak into one of its neighboring peaks. With two variables, we see the solutions in the Peak55 neighborhood decreases, while the solution counts in its neighbor peaks increases (here the neighbor peaks are Peak33 at $x_1 = 0.3$, $x_2 = 0.3$ and Peak35s, which includes both the peak at $x_1 = 0.3$, $x_2 = 0.5$, and the peak at $x_1 = 0.5$, $x_2 = 0.3$).

	# of Top Ranked Solutions in the Neighborhood of:					
σ	Peak 33	Peak 31	Peak 35	Peak 11	Peak 15	Peak 55
0	51	7	327	1	79	528
0.01	29	4	263	3	69	612
0.025	7	3	79	2	31	877
0.05	43	19	144	2	89	675

Table 39: $f_{b_{2var}}$, recommended solution counts with different σ values.

3.5. Summary and Discussion

We have proposed and explored, in the context of benchmark problems for flowshop scheduling, a risk-based concept of robustness for optimization problems. This risk-based concept is distinguishable from the uncertainty-based concept employed in the field of robust optimization. Implementation of our concept requires problem solution methods that sample the solution space intelligently and that produce large numbers of distinct sample points. With these solutions in hand, their robustness scores are easily obtained and heuristically robust solutions found.

We used a conventional genetic algorithm to search for robust solutions, and collected high-quality solutions in a heap during the search process. In our (risk-based sense) approach, the results show little difference with respect to robustness in the recommended solutions for the base and the perturbed scenarios. The best solutions found by the GA on the unperturbed problem are as good as the best solutions found by the GA when trained on perturbed problems generated by the same process as the comparison test problems. A GA could itself be a procedure for finding robust solutions.

To answer the question, we adopt the same core procedure of the proposed approach to search risk-based robust solutions and modify the solution-collecting process. The modification re-directs the proposed approach to searching uncertainty-based robust solutions. For all test problems, with or without the training on perturbed problem, the top solutions found by the GA consist of all local optima (or the near local optima) with fitness values superior to the decision maker's acceptance threshold. The inclusion of all acceptable local optima (near-optima) in the top recommended solution set demonstrates that a GA is itself a procedure for finding robust solutions in either risk- or uncertainty-based sense for all our test problems.

We also examine the possible factors that may affect the performance of proposed GA. Experiment results show that the variable precision, the estimated error size, and the decision-maker's willingness to accept sub-optimal fitness values in exchange for solution robustness all directly influence the search result. We find that training with moderate perturbation improves the GA's efficiency of identifying robust solutions in terms of reducing the number of tracked solutions during the solution collecting process. Proper variable precision also has a similar effect on the GA's search.

By extending the test problems from single-variable to two-variable problems, the effectiveness of GAs in searching for robust solution is accentuated by the reduction on solution tracking. For example, for $f_{b_{1var}}$, one needs to track the top 5% solutions if solutions around all top three local optimum are desired. When solving problem $f_{b_{2var}}$, the GA performs equally well by tracking only the top 0.1% solutions in the two-variable case. Considering that the GA used in our experiments is a rather standard GA with minimum tuning, we believe GA could be highly efficient in providing useful information with regard to searching robust solutions for the optimization problems.

While a GA is surely appropriate for this context, it would seem that any population-based metaheuristic might be used in this way. Systematic investigation of effective means to find robust solutions (in our risk-based sense) has not yet been undertaken. It will be interesting to see whether heuristics for finding optimal solutions perform well as heuristics for finding robust solutions, and vice versa.

CHAPTER 4 : Summary

Conventionally, when given a real life constrained optimization problem, one builds a model for the actual problem first. While trying to find an optimal solution to the model, one should keep in mind the fact that a model is just a model, there will always be a gap between a model and the problem it depicts. The gap between the model and the depicted problem originates from the fact that constraints not based in logic often have coefficient values which are somewhat arbitrarily chosen, and these constraints are adjustable in a certain degree if the objective value can be reasonably improved. Therefore, in the decision making process which is based on modeling, one should remember that there should always be room for alternative solutions.

This study expands the focus beyond just 'getting the optimal solution' when solving a constrained optimization problem to 'finding alternative solutions' for decision making. Given an optimization model, one can define a set of solutions of interest (SoIs) which if well sampled would be valuable for supporting the deliberation. While exact methods do not typically offer much information about the solutions of interest, this study explores the application of population-based metaheuristics (genetic algorithms, to be more specific) on the challenging task of finding SoIs for the given constrained optimization model with test cases cover five different classes of combinatorial optimization problems. Chapter one elaborates on 1.) why expanding the focus of just finding the optimal solution to include the SoIs for a given constrained optimization problem is valuable, and 2) how metaheuristics can be applied for the task of finding SoIs. Chapters two and three explore two different directions on applying the concept of SoI and how to find them – with regard to multi-objective optimization and to solution robustness. The more detailed summaries of this study are described in the rest of this concluding chapter. Table 40 lists the basic information on the test problems used in this study.

Chapter One: Supporting Deliberation for Optimization Problems with the

Chapter	Problem Class	Test Case	Task	Agent	Solution Space
Chapter 1	Generalized Assignment Problem	Beasley OR Lib GAP4 x 5	30	5	9.31×10^{20}
		Beasley OR Lib GAP5 x 5	24	8	4.72×10^{21}
		Beasley OR Lib GAP9 x 5	30	10	10^{30}
		Beasley OR Lib GAP10 x 5	40	10	10^{40}
		Beasley OR Lib GAP11 x 5	50	10	10^{50}
		Beasley OR Lib GAP12 x 5	60	10	10^{60}
	Generalized Quadratic Assignment Problem	Elloumi c1003(A,B,C,D) x 5	10	3	5.90×10^{4}
		Elloumi 2005Aa	20	5	9.54×10^{13}
		Elloumi c2005De	20	5	9.54×10^{13}
		Elloumi 2408(Aa,Ca)	24	8	4.72×10^{21}
		Crossdock	30	12	2.21×10^{23}
Chapter 2	Stable Marriage Problem	Randomly Generated x 25	20	20	2.43×10^{18}
	Couples Problem	Biro	8	8	1.68×10^{7}
Chapter 3	Flowshop Problem	ISHardest	10	3	3.63×10^{6}
		Car1	10	3	3.63×10^{6}
		Car2	11	5	3.99×10^{7}
		Car3	13	4	6.23×10^{9}
		Car4	12	5	4.79×10^{8}
		Taillard x 9	20	10	2.43×10^{18}

Table 40: Test Case Information

Population Based Metaheuristic Approach

To put it simply, solutions of interest can be divided into two categories, the feasible solutions of interest (FoI) and the infeasible solutions of interest (IoI). The FoI consists of feasible solutions which are suboptimal but with attractive resource usage distributions. The IoI consists of infeasible solutions with better objective values but with small violation(s) on resource constraints. While it is always possible to obtain information about shadow prices, reduced costs, etc. for linear programming problems, there is not much information one can gather with the traditional approaches when dealing with problems other than LPs. With thirty generalized assignment benchmark problems (solution space ranging between 10^{20} and 10^{60}), the feasible-infeasible two-population genetic algorithm (FI-2Pop GA) was shown for the problems studied to be very effective in finding the optimal and near-optimal solutions; and the idea about collecting SoIs from the sampled solutions along the process of searching optimal solution has proven to be effective. The proposed approach is further tested with twenty-five generalized quadratic assignment problems and has consistently proven effective in providing SoIs. The provided SoIs offer useful insight information which is otherwise normally unavailable to decision makers.

Chapter Two: Considering Additional Objectives with the Population Based Metaheuristic Approach

Conventionally, stability is used as the sole objective in finding matching solutions for the simple marriage problem and the DAA style approach is the standard way for finding stable matches. While stability is a desired characteristic for obvious reasons, absolute stability is not necessarily required in real-life matching problems since in many cases it is quite difficult for the unsatisfied agent to break away from its current partner and to find itself a better mate. This gives decision makers some leeway in considering other objectives of the matching at the cost of slight instability. Given three other possible objectives such as, social welfare, fairness, and regret, the proposed approach is tested with twenty-five randomly generated simple marriage problems and has shown effective in providing multiple Pareto optimal solutions (either stable or with minimum instability) which dominate (or strongly dominate) the DAA solution(s) when treating the simple marriage matching as multi-objective problem with regard to the three aforementioned aspects. The proposed approach is further tested with a generalized marriage problem, the couples problem, and has proven to be equally effective in offering SoIs, on top of finding the existing stable solution which the DAA style methods fail to find.

Chapter Three: Finding Robust Solutions with the Population Based Meta-heuristic Approach

In this chapter, a risk-based concept of robustness for optimization problems is proposed and explored in the context of flowshop scheduling problems. This risk-based concept is in distinction to, and complements, the uncertainty-based concept employed in the field of robust optimization. The following approach is proposed for obtaining the risk-based robust solution:

1. based on the original given problem, use the given perturbation regime to generate a set of perturbed problems;
2. solve every perturbed problem to optimality, and the collection of optimal solutions constitutes the SoIs;
3. based on the original given problem, using the same perturbation regime to generate a new large set of perturbed problems;
4. evaluate every SoI in the collection with each newly generated perturbed problem;
5. sort the objective values so obtained from best to worst;
6. estimate the robustness of every SoI at level L to be the objective value at the L^{th} decile of the newly perturbed sample set; and
7. at each level L, designate as robust any SoI with a best robustness score.

With fourteen benchmark flowshop scheduling problems, the proposed approach has shown effective on providing risk-based robust solutions for the tested problems. Interestingly, the best solutions found by the GA on the unperturbed problem are as good as the best solutions found by the GA when trained on perturbed problems. Further examinations are carried out with simple one- and two- dimension functions, and the proposed approach has shown that GA is itself a procedure for finding risk-based robust solutions for all tested functions.

Summary

Given an optimization problem, finding the exact optimal solution is always desired, but if the exact solution is not available, heuristically produced solutions are usually acceptable. This study expands the focus beyond the conventional goal of 'getting the optimal solution' to 'finding the alternative solutions' for supporting deliberation. By defining the solution of interest (SoI) and sampling them effectively, one can unveil the model's hidden information which is unavailable otherwise. With ninety-five test cases covering five combinatorial optimization problem classes (generalized assignment problem, generalized quadratic assignment problem, stable marriage problem, couples problem, and flowshop scheduling problem), this study demonstrates how the proposed systematic approach could not only transform the way in which optimization models are used in practice but also empower users by making information for supporting deliberation readily available. While effectively sampling the solutions of interest is challenging, population-based metaheuristics (specifically, two-population genetic algorithm) has proven to perform well in all test cases. Such results serve as a good starting point for further research on supporting deliberation in decision making process. The findings of this study suggest a direction and delineate a new area of optimization research that is worth more exploration. And we can foresee that more and more inspiring findings will be discovered so as to refine and improve the metaheuristics approach and bring them to a wider application.

APPENDIX

Table 41 contains the test results about the combined effect of different estimated error sizes and variable precisions on solution searching.

	Rank Distribution of Top Ranked Solutions in the Neighborhood of:			Ratio of Top 100/1000 Solutions
Variable Value with 3 Decimal Place Precision				
σ value	Peak 0.5	Peak 0.3	Peak 0.1	P05 : P03 : P01
0.001	28 solutions R_U=100	21 solutions R_U=100	13 solutions R_U=100	55 : 25 : 20
0.0025	34 solutions R_U=100	13 solutions R_U=100	7 solutions R_U=100	55 : 25 : 20
0.005	Rk: 1, R_U=100	Rk: 28, R_U=99	Rk: 43 R_U=97	54 : 26 : 20
0.0075	Rk: 1, R_U=100	Rk: 33, R_U=91	Rk: 51 R_U=81	55 : 25 : 20
0.01	Rk: 1, R_U=100	Rk: 35, R_U=84	Rk: 55 R_U=71	55 : 27 : 18
0.025	Rk: 1, R_U=78	Rk: 62, R_U=45	Rk: 98, R_U=35	74 : 25 : 1
0.05	Rk: 1, R_U=45	Rk: 83, R_U=28	Rk: 128, R_U=25	100 : 0 : 0
0.075	Rk: 1, R_U=40	Rk: 58, R_U=27	Not in top 100 solns	78 : 22 : 0
0.1	Rk: 1, R_U=30	Rk: 32, R_U=25	Not in top 100 solns	70 : 30 : 0
Variable Value with 4 Decimal Place Precision				
σ value	Peak 0.5	Peak 0.3	Peak 0.1	P05 : P03 : P01
0.001	Rk: 9, R_U=99	Rk: 1, R_U=100	Rk: 348, R_U=81	495 : 329 : 176
0.0025	Rk: 30, R_U=95	Rk: 1 R_U=98	Rk: 503, R_U=70	508 : 338 : 154
0.005	Rk: 1 R_U=98	Rk: 5, R_U= 94	Rk: 586 R_U=64	539 : 351 : 110
0.0075	Rk: 1 R_U=94	Rk: 40, R_U= 81	Rk: 726 R_U=53	565 : 377 : 58
0.01	Rk: 1, R_U=92	Rk: 151, R_U=70	Rk: 857, R_U=46	625 : 374 : 1
0.025	Rk: 1 R_U=63	Rk: 409 R_U=38	Not in top 1000 solns	771 : 229 : 0
0.05	Rk: 1 R_U= 43	Rk: 603, R_U=24	Not in top 1000 solns	913 : 87 : 0
0.075	Rk: 1, R_U=39	Rk: 328, R_U=25	Not in top 1000 solns	878 : 122 : 0
0.1	Rk: 1, R_U=23	Rk: 5, R_U=21	Not in top 1000 solns	737 : 263 : 0
Variable Value with 5 Decimal Place Precision				
σ value	Peak 0.5	Peak 0.3	Peak 0.1	P05 : P03 : P01
0.001	$Rk: 1, R_U$=85	$Rk: 1, R_U$=85	Rank 968, R_U=45	403 : 594 : 3
0.0025	Rk: 1, R_U=81	Rk:2 , R_U=79	Not in top 1000 solns	347 : 653 : 0
0.005	Rk: 1, R_U=78	Rk: 8 , R_U= 68	Not in top 1000 solns	460 : 540: 0
0.0075	Rk: 1, R_U=76	Rk: 19, R_U=61	Not in top 1000 solns	531 : 469 : 0
0.01	Rk: 1, R_U=76	Rk: 116 , R_U=47	Not in top 1000 solns	658 : 342 : 0
0.025	Rk: 1, R_U=58	Rk: 290, R_U=32	Not in top 1000 solns	915 : 85 : 0
0.05	Rk: 1, R_U=34	Rk: 176, R_U=20	Not in top 1000 solns	978 : 22 : 0
0.075	Rk: 1, R_U=33	Rk: 127, R_U=19	Rk: 999, R_U=13	845 : 154 : 1
0.1	Rk: 1, R_U=23	Rk: 32, R_U=17	Rk: 672, R_U=12	817 : 159 : 3

Table 41: $f_{b_{1var}}$, error size experiment with different variable precision

Table 42 contains the configuration information of Elloumi Constrained Task Assignment Problem (CTAP) c2005De. Tables 43 and 44 contain the configuration information of Elloumi CTAP 2408Aa. Both CTAP c2005De and CTAP 2408Aa are used as benchmark problems for testing the proposed approach on solving GQAPs and the test results are displayed in section 1.9.4. Contents of each configuration table are described as follows:

- Site Capacity Limit Table: specifies the capacity limit of each site.
- Facility Capacity Required Table: specifies the required capacity by each facility.
- Installment Cost Table: specifies the cost for installing a given facility to a given site. Problem c2005De has no installment cost.
- Communication Cost Table: specifies the cost for communication between a pair of facilities if they are not installed at the same site.

1	2	3	4	5
34	32	32	31	40

(a) Site Capacity Limit

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
2	3	5	6	3	4	7	2	6	8	9	2	8	2	8	7	3	6	2	10

(b) Facility Capacity Required

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
1	0	50	93	50	41	75	59	20	37	76	68	17	53	57	38	64	24	20	70	28
2	0	0	25	24	7	52	72	37	98	76	28	26	49	22	76	6	56	54	66	9
3	0	0	0	17	80	49	64	81	86	15	7	30	35	79	35	65	83	85	42	36
4	0	0	0	0	78	35	8	29	74	10	16	58	100	34	29	96	78	90	68	24
5	0	0	0	0	0	22	74	36	50	18	14	54	69	43	90	18	58	75	70	69
6	0	0	0	0	0	0	79	40	77	84	94	77	95	18	60	7	79	66	78	62
7	0	0	0	0	0	0	0	33	21	9	95	56	81	1	88	14	61	80	40	25
8	0	0	0	0	0	0	0	0	38	93	10	82	29	87	62	55	10	9	4	59
9	0	0	0	0	0	0	0	0	0	5	21	63	73	66	41	78	83	12	40	68
10	0	0	0	0	0	0	0	0	0	0	94	23	1	20	58	85	77	2	22	9
11	0	0	0	0	0	0	0	0	0	0	0	82	23	68	25	56	52	24	90	94
12	0	0	0	0	0	0	0	0	0	0	0	0	59	36	70	76	66	48	93	46
13	0	0	0	0	0	0	0	0	0	0	0	0	0	96	52	9	54	10	3	54
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	26	58	52	81	30
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	28	55	89	97	91
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	78	88	95	37
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	70	19	69
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	49	37
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	47

(c) Communication Cost: Facility-1 x Facility

Table 42: Elloumi CTAP c2005De Test Case Configuration (3 Optimal Solutions: 5435)

1	2	3	4	5	6	7	8
29	30	26	33	11	18	3	9

(a) Site Capacity Limit

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
7	1	9	1	7	5	9	7	9	7	1	5	7	3	4	5	6	2	4	7	4	6	7	10

(b) Facility Capacity Required

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	31	94	17	83	34	2	72	89	7	70	76	24	80	41	80	0	70	68	75	93	69	98	46	2
2	27	44	2	42	86	80	62	8	38	60	25	37	32	91	50	53	2	55	25	25	35	55	95	8
3	43	35	94	35	18	78	75	59	31	89	40	30	69	97	48	44	44	76	5	75	11	35	19	15
4	67	99	49	13	62	14	46	24	32	60	59	60	10	77	75	23	61	5	47	93	48	20	13	28
5	22	68	51	62	78	83	42	79	97	74	41	91	38	8	54	12	49	68	0	50	96	61	47	9
6	47	65	34	99	94	95	27	23	23	50	42	3	41	31	58	89	3	89	85	68	85	35	49	33
7	91	92	90	56	1	17	57	26	6	54	10	66	8	71	24	33	16	97	42	3	1	44	1	20
8	77	22	46	86	74	88	69	77	21	18	30	0	99	72	21	82	20	46	58	36	2	99	0	48

(c) Installment Cost: Site x Facility

Table 43: Elloumi CTAP 2408Aa Test Case Configuration (a) (Optimal Solution: 5643)

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24
1	0	0	0	40	0	16	0	39	0	0	0	0	89	11	5	13	0	0	0	90	0	9	0	46
2	0	0	0	22	0	0	28	28	43	0	44	87	0	55	71	91	0	0	0	0	7	0	61	10
3	0	0	0	0	45	88	0	0	0	43	37	0	0	93	97	99	77	0	39	0	40	0	0	62
4	0	0	0	0	16	0	51	0	0	83	97	95	0	37	88	0	0	49	0	61	0	0	43	78
5	0	0	0	0	0	0	100	90	0	0	47	38	15	0	0	78	4	0	0	0	0	19	35	0
6	0	0	0	0	0	0	0	40	1	0	18	0	0	0	0	0	75	58	0	70	94	0	75	0
7	0	0	0	0	0	0	0	0	44	97	0	0	65	59	0	0	38	9	58	0	66	64	44	0
8	0	0	0	0	0	0	0	0	0	0	30	0	0	36	52	0	0	0	71	0	0	61	0	0
9	0	0	0	0	0	0	0	0	0	0	0	16	76	37	91	27	96	0	0	89	12	90	11	90
10	0	0	0	0	0	0	0	0	0	0	42	57	0	81	0	0	0	92	65	99	0	24	57	0
11	0	0	0	0	0	0	0	0	0	0	0	17	97	0	0	28	95	65	0	0	87	0	20	13
12	0	0	0	0	0	0	0	0	0	0	0	0	83	0	38	0	55	0	0	74	0	0	45	77
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	40	0	0	0	52	0	33	34	63	94
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	52	0	69	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	92	98	0	72	0	60	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	4	0	7	63	0	0	86	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	38	60	64	0	77	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	63	0	53	57
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	58	79
21	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	6	55	0
22	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	49	0
23	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	27

(a) Communication Cost: (Facility-1) x Facility

Table 44: Elloumi CTAP 2408Aa Test Case Configuration (b) (Optimal Solution: 5643)

BIBLIOGRAPHY

R. K. Ahuja, J. B. Orlin, and A. Tiwari. A greedy genetic algorithm for the quadratic assignment problem. *Computers and Operations Research*, 27:917–934, 2000.

J. T. Alander. *An indexed bibliography of genetic algorithms: Years 1957-1993*, volume II of *Practical Handbook of GENETIC ALGORITHMS*. New Frontiers, Cambridge, Massachusetts, 1995. ISBN 0-262-08213-6 (hc) / ISBN 0- 262-58111-6 (pbk.).

B. Aldershof and O. M. Carducci. Stable matching with couples. *Discrete Applied Mathematics*, 68:203–207, 1996.

B. Aldershof and O. M. Carducci. Stable marriage and genetic algorithms: A fertile union. *Journal of Heuristics*, 5:29–46, 1999.

R. L. Axtell and S. O. Kimbrough. How much social welfare should be sacrificed in the pursuit of stability? In *Proceedings of the 2008 World Congress on Social Simulation (WCSS-08)*, 2008.

D. Bai, T. Carpenter, and J. Mulvey. Making a case for robust optimization models. *Management Science*, 43(7):895–907, 1997. Publisher: INFORMS.

J. E. Beasley. Or-library. World Wide Web, Accessed July 27, 2009. `http://people.brunel.ac.uk/~mastjjb/jeb/info.html`, 2009.

A. Ben-Tal and A. Nemirovski. Robust solutions to uncertain programs. *Operations Research Letter*, 25:1–13, 1998.

A. Ben-Tal and A. Nemirovski. Robust solutions of linear programming problems contaminated with uncertain data. *Mathematical Programming*, 88:411–424, 2000.

A. Ben-Tal and A. Nemirovski. Robust optimization - methodology and applications. *Mathematical Programming*, 92:453–482, 2002.

D. Bertsimas and M. Sim. The price of robustness. *Operations Research*, 52(1):35–53, Jan-Feb 2004.

D. Bertsimas and A. Thiele. Robust and data-driven optimization; modern decision-making under uncertainty. *Tutorials in Operations Research, INFORMS*, 2006a.

D. Bertsimas and A. Thiele. A robust optimization approach to inventory theory. *Operations Research*, 54(1):150–168, 2006b.

H.-G. Beyer and B. Sendhoff. Robust optimization - a comprehensive survey. *Computer Methods in Applied Mechanics and Engineering*, 196:3190–3218, 2007.

P. Biro, R. W. Irving, and I. Schlotter. Stable matching with couples - an empirical study. *Journal of Experimental Algorithmics*, 16, 2011. Article No.:1.2.

J. Branke. Creating robust solutions by means of evolutionary algorithms. *Parallel Problem Solving from Nature V. LNCS*, 1498:119–128, 1998.

B. Branley, R. Fradin, S. Kimbrough, and T. Shafer. On heuristic mapping of decision surfaces for post-evaluation analysis. In R. H. J. Sprague, editor, *Proceedings of the Thirtieth Annual Hawaii International Conference on System Sciences*, Los Alamitos, CA, 1997. IEEE Press.

E. K. Burke, E. Hart, G. Kendall, J. Newall, P. Ross, and S. Shulenburg. An emerging direction in modern search technology. In F. Glover and G. Kochengerger, editors, *Handbook of Metaheuristics*, volume II, page 457474. Kluwer, Cambridge, Massachusetts, 2003.

J. Carlier. Ordonnancements a contraintes disjonctives. *R.A.I.R.O. Recherche Operationelle / Operations Research*, 12:333–351, 1978.

CaRMS. Canadian resident matching service. http://www.carms.ca/. Last access, Mar. 1, 2014.

K. Cechlarova and S. Ferkova. The stable crews problem. *Discrete Appl. Math.*, 140(1–3): 1–17, 2004.

K. Cechlarova and T. Fleiner. On a generalization of the stable roommates problem. *ACM Transactions on Algorithms*, 1(1):143–156, July 2005.

R. Cheng, M. Gen, and Y. Tsujimura. A tutorial survey of job-shop scheduling problems using genetic algorithms - part i. *Computers and Industrial Engineering*, 30(4):983–997, 1996.

R. Cheng, M. Gen, and Y. Tsujimura. A tutorial survey of job-shop scheduling problems using genetic algorithms - part ii. *Computers and Industrial Engineering*, 36(2):343–364, 1999.

P. Chu and J. Beasley. A genetic algorithm for the generalized assignment problem. *Computers and Operations Research*, 24(1):17–23, 1997.

C. Coello. Theoretical and numerical constraint handling techniques used with evolutionary algorithms: A survey of the state of the art. *Computer Methods in Applied Mechanics and Engineering*, 191:1245–1287, 2002.

C. Coello. List of references on constraint-handling techniques used with evolutionary algorithms. `http://www.cs.cinvestav.mx/~constraint/` (Accessed Mar. 1st, 2013), 2008.

C. Coello and E. Montes. Constraint-handling in genetic algorithms through the use of dominance-based tournament selection. *Advanced Engineering Informatics*, 16(3):193–203, July 2002.

C. A. Coello Coello, G. B. Lamont, and D. A. Van Veldhuizen. *Evolutionary Algorithms for Solving Multi-Objective Problems*. Springer science + Business Media LLC, New York, NY, 2 edition, 2007.

J.-F. Cordeau, M. Gaudioso, G. Laporte, and L. Moccia. A memetic heuristic for the generalized quadratic assignment problem. *INFORMS Journal on Computing*, 18(4): 433–443, 2006.

D. Dasgupta, G. Hernandez, D. Garrett, P. K. Vejandla, A. Kaushal, R. Yerneni, and J. Simien. A comparison of multiobjective evolutionary algorithms with informed initialization and Kuhn-Munkres algorithms for the sailor assignment problem. In *Proceedings of the 2008 GECCO Conference Companion on Genetic and Evolutionary Computation*, pages 2129–2134, New York, NY, USA, 2008. ACM.

K. Deb. *Multi-Objective Optimization Using Evolutionary Algorithms*. John Wiley and Sons, LTD, Chichester, UK, 2001.

K. Deb and S. Agarwal. A niched-penalty approach for constraint handling in genetic algorithms. In *Proceedings of the ICANNGA*, Portoroz, Slovenia, 1999. Morgan Kaufmann Publishers Inc.

J. Diaz and E. Fernandez. A tabu search heuristic for the generalized assignment problem. *European Journal of Operational Research*, 132:2238, 2001.

C. M. D.O. Boyer and J. Perez. Algorithm with crossover based on confidence intervals as an alternative to least squares estimation for nonlinear models. MIC 2001, 4th Metaheuristics International Conference, 2001.

Z. Drezner. A new genetic algorithm for the quadratic assignment problem. *INFORMS Journal on Computing*, 15(3):320–330, 2003.

K. Echenique and M. B. Yenmez. A solution to matching with preferences over colleagues. *Games and Economic Behavior*, 59(1):46–71, 2007.

S. Elloumi. Tapsite. World Wide Web, Accessed Feb 27, 2014. cedric.cnam.fr/oc/TAP/TAP.html, 2014.

V. Estivill-Castro. The role of selection in genetic algorithms. In F. Glover and G. Kochengerger, editors, *Technical Report Queensland University of Technology Faculty of Information Technology*, volume 4-96. Queensland University of Technology, Cambridge, Massachusetts, 1996. ISBN 1402072635.

J. E. Falk. Exact solutions of inexact linear programs. *Operations Research*, 24(4):783–787, 1976. published by INFORMS.

H. Feltl and G. Raidl. An improved hybrid genetic algorithm for the generalized assignment problem. In *Proceedings of the ACM Symposium on Applied Computing (SAC04)*, pages 990–995, New York, NY, USA, 2004. ACM. ISBN:1–58113–812–1.

S. G. Ficici. Multiobjective optimization and coevolution. In J. Knowles, D. Corne, and K. Edb, editors, *Multiobjective Problem Solving from Nature: From Concepts to Applications*, Natural Computing, pages 31–52. Springer, Berlin, Germany, 2008.

J. M. Fitzpatrick and J. J. Grefenstette. Genetic algorithms in noisy environments. *Machine Learning*, 3:101–120, 1988.

G. W. Flake. *The Computational Beauty of Nature.* A Bradford Book. The MIT Press, Cambridge, Massachusetts, 1998. ISBN-13 978-0-262-06200-8 (hc.) / 978-0-262- 56127-3 (pbk.).

T. Fleiner, R. W. Irving, and D. F. Manlove. Efficient algorithms for generalized stable marriage and roommates problems. *Theoretical Computer Science archive*, 381(1–3): 162–176, Aug. 2007.

T. Fuku, A. Namatame, and T. Kaizouji. Collective efficiency in two-sided matching. In P. Mathieu, B. Beaufils, and O. Brandouy, editors, *Artificial Economics; Agent-Based Methods in Finance, Game Theory and Their Applications*, Lecture Notes in Economics and Mathematical Systems, pages 115–126. Springer-Verlag, Berlin, Germany, 2006.

D. Gale and L. S. Shapley. College admissions and the stability of marriage. *The American Mathematical Monthly*, 69(1):9–15, Jan. 1962.

GAMSWorld. Gams world. global world. Web site http://www.gamsworld.org/global/index.htm (Accessed Dec. 16, 2013), 2013.

A. M. Geoffrion and R. Nauss. Parametric and postoptimality analysis in integer linear programming. *Management Science*, 23(5):453–466, January 1977.

L. E. Ghaoui and H. Lebret. Robust solutions to least-squares problems with uncertain data matrices. *SIAM Journal on Matrix Analysis and Applications*, 18:1035–1064, 1997.

F. Glover. Genetic algorithms and scatter search: unsuspected potentials. *Statistics and Computing*, 4:131–140, 1994.

F. Glover and M. Laguna. Fundamentals of scatter search and path relinking. *Control and Cybernetics*, 29(3):653–684, 2000.

F. W. Glover and G. A. Kochenberger, editors. *Handbook of Metaheuristics.* International Series in Operations Research & Management Science. Springer, Cambridge, Massachusetts, 2003. ISBN 1402072635.

D. E. Goldberg. *Genetic Algorithms in Search, Optimization & Machine Learning.* Addison-Wesley Publishing Company, Inc., Reading,MA, 1989.

D. Goldfarb and G. Iyengar. Robust portfolio selection problems. *Mathematics of Operations Research*, 28(1):1–38, February 2003.

H. J. Greenberg. How to analyze the results of linear programspart 1: Preliminaries. *Interfaces*, 23(4):56–67, July–August 1993a.

H. J. Greenberg. How to analyze the results of linear programspart 2: Price interpretation. *Interfaces*, 23(5):97–114, September–October 1993b.

H. J. Greenberg. How to analyze the results of linear programspart 3: Infeasibility diagnosis. *Interfaces*, 23(6):120–139, November–December 1993c.

H. J. Greenberg. How to analyze the results of linear programspart 1: Forcing substructures. *Interfaces*, 24(1):121–130, January–February 1994.

H. J. Greenberg. *An Annotated Bibliography for Post-Solution Analysis in Mixed Integer Programming and Combinatorial Optimization*, volume 108 of *UCD/CCM report*. University of Colorado at Denver, Center for Computational Mathematics, 1997.

D. Gusfield and R. W. Irving. *The Stable Marriage Problem: Structure and Algorithms*. MIT Press, Cambridge, MA, 1989. ISBN 978-0262515528.

P. M. Hahn, B.-J. Kim, M. Guignard, J. M. Smith, and Y.-R. Zhu. An algorithm for the generalized quadratic assignment problem. *Computational Optimization and Applications*, 40(3):351–372, 2008.

M. M. Halldorsson, K. Iwama, S. Miyazaki, and H. Yanagisawa. Improved approximation results for the stable marriage problem. *ACM Transactions on Algorithms*, 3(30), July 2007.

E. Hart, P. Ross, and D. Corne. Evolutionary scheduling: A review. *Genetic Programming and Evolvable Machines*, 6(2):191–220, 2005.

S. R. Hejazi and S. Saghafian. Slowshop-scheduling problems with makespan criterion: A review. international journal of production research. *International Journal of Production Research*, 43(14):2895–2929, July 2005.

J. W. Herrmann. A genetic algorithm for minimax optimization problems. In *Proceedings of the Congress on Evolutionary Comptation*, volume 2, pages 1099–1103. IEEE, 1999.

J. H. Holland. *Adaptation in Natural and Artificial Systems*. A Bradford Book. The MIT Press, Cambridge, Massachusetts, 1998. ISBN 0-262-08213-6 (hc) / ISBN 0- 262-58111-6 (pbk.).

E. Ignall and L. E. Schrage. Application of the branch-and-bound technique to some flowshop problems. *Operations Research*, 13:400–412, 1965.

R. Irving. Stable marriage and indifference. *Discrete Applied Mathematics*, 48:261–272, 1994.

R. Irving. Scottish prho allocations. `http://www.dcs.gla.ac.uk/~rwi/SPA.html`. Last access, Mar. 1, 2014.

R. Irving and D. Manlove. The stable roommates problem with ties. *Journal of Algorithm*, 43(1):85–105, 2002.

R. W. Irving and P. Leather. The complexity of counting stable marriages. *Society for industrial and Applied Mathematics (SIAM) Journal of Computation*, 15(3), Aug. 1986.

R. W. Irving and S. Scot. The stable fixtures problem - a many-to-many extension of stable roommates. *Discrete Applied Mathematics*, 155:2118–2129, 2007.

K. Iwama and S. Miyazaki. A survey of the stable marriage problem and its variants. In *Proceedings of International Conference on Informatics Education and Research for Knowledge-Circulating Society (ICSK 2008)*, 10662 Los Vaqueros Circle, P.O. Box 3014, Los Alamitos, CA 90720, 2008. IEEE Computer Society. ISBN 0-7695-3128-8.

K. Iwama, D. Manlove, S. Miyazaki, and Y. Morita. Stable marriage with incomplete lists and ties. *Proceedings of ICALP 1999, the 26th International Colloquium on Automata, Languages and Programming, Lecture Notes in Computer Science, Prague, Czech Republic*, 1644:443–452, July 1999.

M. T. Jensen. Generating robust and flexible job shop schedules using genetic algorithms. *IEEE Transactions on Evolutionary Computation*, 7(3):275–288, June 2003.

I. John J. Bartholdi and K. R. Gue. The best shape for a crossdock. *INFORMS Transportation Science*, 38(2):235–244, May 2004. Article No.:1.2.

K. Jong and W. Spears. *A formal Analysis of the Role of Multi-point Crossover in Genetic Algorithms*, volume 5 of *Annals of Mathematics and Artificial Intelligence*. Springer, Netherlands, 1992. pp.1–26.

JRMP. Japan residency matching program. `http://www.carms.ca/`. Last access, Mar. 1, 2014. Site temporarily out of service and scheduled to resume from Mar. 17, 2014.

J.Shoaf and J.A.Foster. The efficient set ga for stock portfolios. In *The IEEE International Conference on Evolutionary Computation Proceedings*, 1998.

H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems*. Springer, Berlin, 2004.

S. Kimbrough and D.H.Wood. On how solution populations can guide revision of model parameters. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2006)*, Seattle, WA, USA, 2006. ACM. July 8-12, 2006; Late breaking papers.

S. Kimbrough, M. Lu, and D. Wood. Exploring the evolutionary details of a feasible-infeasible two-population ga. In *Proceedings of the 8th International Conference on Parallel Problem Solving from Nature (PPSN VIII)*, pages 415–421, Birmingham, UK, 2002a. Morgan Kaufmann. 18-22 September, 2004.

S. Kimbrough, M. Lu, D. Wood, and D. Wu. Exploring a two-market genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2002)*, pages 415–421, San Francisco, CA, 2002b. Morgan Kaufmann.

S. Kimbrough, M. Lu, D. Wood, and D. Wu. Exploring a two-population genetic algorithm. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO 2003)*, pages 1148–1159, , Berlin, Germany, 2002c. Springer. LNCS, vol2723.

S. Kimbrough, M. Lu, and S. Safavi. Exploring a financial product model with a two-population genetic algorithm. In *Proceedings of the 2004 Congress on Evolutionary Computation*, pages 855–862, Piscataway, NJ,, 2004. IEEE Neural Network Society, IEEE Service Center. ISBN 0-7803-8515-2.

S. Kimbrough, G. Koehler, M. Lu, and D. Wood. On a feasible-infeasible two-population genetic algorithm for constrained optimization: Distance tracing and no free lunch. *European Journal of Operational Research*, 190:310–327, 2008.

S. Kimbrough, A. Kuo, H. Lau, Lindawati, and D. Wood. On using genetic algorithms to support post-solution deliberation in the generalized assignment problem. MIC 2009: The VIII METAHEURISTICS INTERNATIONAL CONFERENCE, conference CD, July 2009a. 13–16 July 2009.

S. O. Kimbrough, A. Kuo, H. C. Lau, Lindawati, and D. H. Wood. On using genetic algorithms to support post-solution deliberation in the generalized assignment problem. MIC2009, The VIII Metaheuristics International Conference, 2009b. 2009, July 13–16, Hamburg, Germany.

S. O. Kimbrough, A. Kuo, and H. C. Lau. Effective heuristic methods for finding non-optimal solutions of interest in constrained optimization models. In *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO-2010)*. ACM, 2010.

M. Kirschner and J. Gerhart. Evolvability. In *Proceedings of the National Academy of Sciences of the United States of America*, volume 95, pages 8420–8427. PNAS, 1998. number 15.

B. Klaus, F. Klijn, and J. Masso. Some things couples always wanted to know about stable matchings (but were afraid to ask). *Review of Economic Design*, 11(3):175–184, Nov 2007.

D. E. Knuth. *Stable Marriage and Its Relation to Other Combinatorial Problems: An Introduction to the Mathematical Analysis of Algorithms*, volume 10 of *CRM Proceedings and Lecture Notes, Centre de Recherches Mathématiques Université de Montréal*. American Mathematical Society, Providence, RI, 1997. Originally published as Knuth1976.

F. Kojima. School choice: Impssibilities for affirmative action. *Games and Economic Behavior*, 75:685–693, 2012.

T. Lau and E. Tsang. The guided genetic algorithm and its application to the generalized assignment problem. Tenth IEEE International Conference on Tools with Artificial Intelligence, Nov 1998. 1998,10-12, Nov. pp.336-343.

S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations.* John Wiley & Sons, New York, NY, 1990.

D. G. McVitie and L. B. Wilson. The stable marriage problem. *Communications of the ACM archive*, 14(7):486–490, July 1971a.

D. G. McVitie and L. B. Wilson. Algorithm 411: Three procedures for the stable marriage problem. *Communications of the ACM archive*, 14(7):491–492, July 1971b.

Z. Michalewicz. A survey of constraint handling techniques in evolutionary computation methods. In *Proceedings of the 4th Annual Conference on Evolutionary Programming*, pages 135–155, Cambridge, MA, 1995. MIT Press.

Z. Michalewicz. *Genetic Algorithms + Data Structures = Evolutaion Programs*, volume II of *Practical Handbook of GENETIC ALGORITHMS.* Springer, Berlin, Germany, third edition, 1996. ISBN 0-262-08213-6 (hc) / ISBN 0- 262-58111-6 (pbk.).

Z. Michalewicz and D. Fogel. *How to Solve It: Modern Heuristics*, volume II of *Practical Handbook of GENETIC ALGORITHMS.* Springer, Berlin, Germany, 2000. ISBN 0-262-08213-6 (hc) / ISBN 0- 262-58111-6 (pbk.).

B. L. Miller and D. E. Goldberg. Genetic algorithms, selection scheme, and the varying effect of noise. *Evolutionary Computation*, 4(2):113–131, 1996.

O. Morgenstern. *On the Accuracy of Economic Observations.* Princeton University Press, Princeton, New Jersey, 1963.

J. M. Mulvey, R. J. Vanderbei, and S. A. Zenios. Robust optimization of large-scale systems. *Operations Research*, 43(2):264–281, Mar-Apr 1995.

M. Nakamura, K. Onaga, S. Kyan, and M. Silva. Genetic algorithm for sex-fair stable marriage problem. In *Circuits and Systems, ISCAS '95., IEEE International Symposium on*, volume 1, pages 509–512, Apr 1995.

R. Nauss. Solving the generalized assignment problem: An optimizing and heuristic approach. *Informs Journal on Computing*, 15:249266, 2003.

NRMP. National resident matching program. `http://www.nrmp.org/`. Last access, Mar. 1, 2014.

E. Peranson and R. R. Randlett. The nrmp matching algorithm revisited: Theory versus practice. *Acdemic Medicine*, 70(6):477–484, June 1995.

J. Pfeiffer. *Combinatorial Auctions and Knapsack Problems - An Analysis of Optimization Methods*, volume II. VDM Verlag Dr. Mueller e.K., LaVergne, Tennessee, USA, 2007. ISBN 978-3-8364-5006-5.

M. L. Pinedo. *Scheduling: Theory, Algorithms, and Systems.* Springer, Berlin, Germany, 3 edition, 2008.

C. R. Reeves. A genetic algorithm for flowshop sequencing. *Computers and Operations Research*, 22(1):5–13, 1995.

J. R.J. Bauer. *Genetic Algorithms and Investment Strategies*, volume II of *Practical Handbook of GENETIC ALGORITHMS.* John Wiley & Sons, Inc., 1994. ISBN 0-471-57679-4.

A. E. Roth. The evolution of the labor market for medical interns and residents: A case study in game theory. *Journal of Political Economy*, 92(6):991–1016, 1984.

A. E. Roth. New physicians: A natural experiment in market organization. *Science*, 250: 1524–1528, 1990.

A. E. Roth. Deferred acceptance algorithms: History, theory, practice, and open questions. *International Journal of Game Theory*, 36:537–569, 2008. DOI 10.1007/s00182-008-0117-6.

A. E. Roth and E. Peranson. The redesign of the matching market for american physicians: Some engineering aspects of economic design. *American Economic Review*, 89(4):748–780, September 1999.

A. E. Roth and M. A. Sotomayor. *Two-Sided Matching: A Study in Game-Theoretic Modeling and Analysis.* Econometric Society Monographs. Cambridge University Press, Cambridge, UK, 1990.

M. Savelsbergh. A branch-and-price algorithm for the generalized assignment problem. *Operations Research*, 45:831841, 1997.

C. Singh. Convex programming with set-inclusive constraints and its applications to generalized linear and fractional programming. *Journal of Optimization Theory and Applications*, 38(1):33–42, 1982.

A. L. Soyster. Convex programming with set-inclusive constraints and applications to inexact linear programming. *Operations Research*, 21(5):1154–1157, 1973. published by INFORMS.

W. Spears. Adapting crossover in a genetic algorithm. Naval Research Laboratory AI Center Report AIC-92-025, 1992. Washington, DC.

G. Syswerda. Uniform crossover in genetic algorithms. In J. D. Schaffer, editor, *Proceedings of the third International Conference in Genetic Algorithms*, San Francisco, CA, USA, 1989. ACM, Morgan Kaufmann Publishers Inc.

G. Taguchi. *Quality Engineering through Design Optimization.* Kraus International Publications, 1984.

G. Taguchi. *Introduction to Quality Engineering.* Kraus International Publications, 1986.

E. Taillard. Eric taillard's page. `http://mistic.heig-vd.ch/taillard/problemes.dir/ordonnancement.dir/ordonnancement.html`. Access 20130701.

D. Tate and A. Smith. Expected allele coverage and the role of mutation in genetic algorithms. In *Proceedings of the Fifth International Conference on Genetic Algorithms*, 1993.

S. Tsutsui and A. Ghosh. Genetic algorithms with a robust solution searching scheme. *IEEE Transactions on Evolutionary Computation*, 1(3):201–208, September 1997.

N. A. Vien and T. C. Chung. Multiobjective fitness functions for stable marriage problem using genetic algorithm. SICE-ICASE, International Joint Conference, October 2006.

N. A. Vien, N. H. Viet, H. Kim, S. Lee, and T. Chung. Ant colony based algorithm for stable marriage problem. In K. Elleithy, editor, *Advances and Innovations in Systems Computing Sciences and Software Engineering*, pages 457–461. Springer, Cordrecht, The Netherlands, 2007. ISBN 978-1-4020-6263-6 (HB), 978-1-4020-6264-3 (e-book).

C. Voudouris and E. P. Tsang. Guided local search. In F. Glover and G. Kochengerger, editors, *Handbook of Metaheuristics*, pages 185–218. Kluwer, Cambridge, Massachusetts, 2003.

A. Wagner. *Robustness and Evolvability in Living Systems.* Princeton Studies in Complexity. Princeton University Press, Princeton, NJ, 2005.

K. J. Williams. A reexamination of the nrmp matching algorithm. *Acdemic Medicine*, 70 (6):470–476, June 1995.

J. Wilson. A genetic algorithm for the generalised assignment problem. *Journal of the Operational Research Society*, 48:804–809, July–August 1993.

D. Wolpert and W. Marcready. No free lunch theorems for optimization. *IEEE Transactions on Evolutionary Computation*, 1(1), April 1997.

M. Yagiura, T. T. Ibaraki, and F. Glover. A path relinking approach with ejection chains for the generalized assignment problem. *European Journal of Operational Research*, 169: 548569, 2006.

O. Yeniay. Penalty function methods for constrained optimization with genetic algorithms. *Mathematical and Computational Applications*, 10(1):45–56, 2005.