Strategic Solution Combination in Scatter Search for Quadratic Unconstrained Binary Optimization

Justin Pauckert[1,2], Matthieu Parizy[1][a] and Mayowa Ayodele[3][b]
[1]*Fujitsu Ltd, Kawasaki, Japan*
[2]*Technical University of Berlin, Berlin, Germany*
[3]*Fujitsu Research of Europe, Slough, U.K.*

Keywords: QUBO, Scatter Search, Path Relinking, Tabu Search.

Abstract: Quadratic Unconstrained Binary Optimization (QUBO) has become an important unifying model for formulating combinatorial optimization problems. Since QUBO is NP-hard, it is common to apply heuristics to them. Although a wider range of heuristics have been applied to QUBOs in the past, high performing QUBO solvers using specialized hardware have been of interest in more recent years. These solvers use local search algorithms such as Simulated Annealing, Quantum Annealing and Tabu Search to improve solution quality, but few also include global search methods to explore the solution space. Since research around global search for QUBO solvers is scarce, in this work we compare different variants implemented within a framework known as scatter search. We compare two global search methods, both used to generate new solutions from known solutions. The first method is path relinking which has been shown to deliver the best results in the literature for hard QUBO problems when used in scatter search. The second method is uniform crossover and is widely used in the field of Evolutionary Algorithms (EAs). We show that better performance can be achieved using path relinking compared to uniform crossover on knapsack problem instances.

1 INTRODUCTION

Methods for solving combinatorial optimization problems have been of interest for many decades. These problems occur in many real-world scenarios such as vehicle routing (Toth and Vigo, 2001), synthetic biology (Naseri and Koffas, 2020) and production scheduling (Moreno et al., 2010). In classical operations research, many problem representations (e.g. permutation, binary and integer vector) are used to solve these problems. In more recent years, Quadratic Unconstrained Binary Optimization (QUBO) problems have become a widely used model for representing well-known combinatorial optimization problems to make techniques from related fields applicable. They are defined as

$$\min_{x\in\{0,1\}^n} x^T Q x = \sum_{i=1}^{n}\sum_{j=1}^{i} q_{ij}x_i x_j \qquad (1)$$

where $Q \in \mathbb{R}^{n\times n}$ is a symmetric matrix. There exist embeddings for many classical combinatorial optimization problems (Lucas, 2014). Because of their similarity to the Ising Model (Cipra, 1987), QUBO formulations are often used to solve problems on Ising machines such as D-Wave's Quantum Annealer (Johnson et al., 2011) and Fujitsu's Digital Annealer (Hiroshi et al., 2021). These solvers can outperform many classical algorithms implemented on general-purpose computers (Matsubara et al., 2020; Aramon et al., 2019; Ayodele, 2022). Given their different architecture, appropriate algorithms are needed to utilize their unique properties.

This work contributes to research focused on creating effective algorithms for combinatorial optimization problems represented as QUBO. We make use of scatter search which has shown to present promising performance (Samorani et al., 2019). Two variations of it are compared to each other. One uses path relinking, the other employs uniform crossover. Analysis is done using knapsack problem instances. Time-to-solution is considered for small problems. Solution quality after a fixed time limit is taken into account for larger instances.

The rest of this paper is structured as follows. A review of literature is presented in Section 2. The problem formulations are presented in Section 3. We describe the algorithms used in this study in Section 4. Used problem sets and experimental settings are pre-

[a] https://orcid.org/0000-0002-5777-7756
[b] https://orcid.org/0000-0003-0854-4777

Pauckert, J., Parizy, M. and Ayodele, M.
Strategic Solution Combination in Scatter Search for Quadratic Unconstrained Binary Optimization.
DOI: 10.5220/0011547600003332
In *Proceedings of the 14th International Joint Conference on Computational Intelligence (IJCCI 2022)*, pages 133-142
ISBN: 978-989-758-611-8; ISSN: 2184-2825

sented in Section 5. Results are analyzed in Section 6. Finally, conclusions and further work are discussed in Section 7.

2 PREVIOUS WORK

2.1 Fast Local Neighborhood Evaluation

First of all, cost evaluations of single flip neighbours of a solution can be simplified. That is, given a current solution $x \in \{0,1\}^n$, the cost of every $x' \in \{0,1\}^n$ which differs from x in exactly one entry can be evaluated without computing $x'^T Q x'$ for every neighbor (Glover and Hao, 2010). If we know the current cost $x^T Q x$, we can efficiently compute the cost of any single flip neighbour $x' \in N(x)$. Let x and x' differ at index $i \in \{1,...,n\}$, then $E(x') = x^T Q x + \Delta_i$ where Δ_i is given by

$$\Delta_i = (1-2x_i) \cdot q_{ii} + \sum_{\substack{j=1, j\neq i \\ x_j=1}}^{n} q_{ij}. \qquad (2)$$

More recently neighborhood evaluation was improved with hardware-accelerated algorithms. GPUs can be used in parallel to reduce the computational cost for neighborhood evaluation to $O(1)$ (Yasudo et al., 2020). Fujitsu's Digital Annealer uses custom CMOS hardware to consider a flip of each variable in parallel and thereby increase the acceptance rate of new solutions in the simulated annealing process (Aramon et al., 2019).

2.2 Metaheuristics for QUBO Problems

Although exact solving methods for QUBOs exist, they are often unsuitable for large instances. Instead, heuristic methods have been developed to find good solutions within a reasonable amount of computation time (Kochenberger et al., 2014). Most popular methods tend to utilize either Simulated Annealing (SA) (Alkhamis et al., 1998) or Tabu Search (TS) (Glover et al., 1998) as local search technique. Based on that, you can distinguish between local search based methods (e.g. Multi-Start Tabu Search (Palubeckis, 2004)) and population based ones (e.g. scatter search (Amini et al., 1999), Memetic algorithms (Merz and Katayama, 2004)). An extensive comparison of the most popular heuristic algorithms for QUBO has been conducted in (Dunning et al., 2018).

Heuristics are designed to find good-enough feasible solutions within a limited amount of time when the use of an exact solver is impractical because of the complexity of the problem. Metaheuristics are more general algorithms that can be applied to a broader range of problems. They provide a high-level framework to combine strategies and allow for a flexible foundation to find good optimization algorithms (Sorensen et al., 2017). Depending on the class of problem that is to be solved, metaheuristics can be tailored to obtain solutions as fast as possible. Metaheuristics applied to QUBO formulations of combinatorial optimization problems include Ant Colony Optimization (ACO) (Cao et al., 2018), Differential Evolution (DE) (Deng et al., 2020), Estimation of Distribution Algorithm (EDA) (Soloviev et al., 2021), Genetic Algorithm (GA) (Supasil et al., 2021), Particle Swarm Optimization (PSO) (Fujimoto and Nanai, 2021), and scatter search (Samorani et al., 2019). Scatter search has been shown to produce competitive results for QUBO problems in (Samorani et al., 2019; Wang et al., 2012), which is why we use it in this study. Like many other evolutionary algorithms, scatter search uses the concept of creating an offspring solution from a set of parent solutions. These mechanisms are referred to as crossover methods or combination methods. They are particularly used in GA, DE and scatter search.

Uniform Crossover. Uniform crossover has been reported as one of the most successful combination methods for GAs (Picek et al., 2011; Umbarkar and Sheth, 2015). It is still recognized as one of the most powerful crossover operators in recent publications (Zainuddin et al., 2020). Although it has also been used within the scatter search framework before (Hakli and Ortacay, 2019), its effectiveness for QUBO problems in that setting is yet to be assessed.

Path Relinking. Path relinking is another combination method and has been successfully employed with scatter search applied to QUBO problems. Different variations of path relinking were used in (Wang et al., 2012), namely greedy path relinking and random path relinking. They only differ slightly and both were shown to have a similar impact on performance so we will restrict ourselves to greedy path relinking for the purpose of this paper and only refer to it as path relinking. The impact of parent selection, that is, which solutions to combine through path relinking, has been studied by some. A more sophisticated parent selection based on clusters in the reference set was used in (Samorani et al., 2019) to prevent redundant combinations which produce similar offspring. A hy-

brid approach in (Glover and Hao, 2010) utilized Tabu Search within the evolutionary algorithm framework and chose parents to perform path relinking based on their quality and distance.

Both methods generate children which are similar to their parents. Path relinking relies on multiple function evaluations for each step. Uniform crossover randomizes bits and is therefore able to combine solutions much faster. Given the reliable performance of path relinking with scatter search for QUBO problems as well as the competitive performance of uniform crossover in evolutionary algorithms applied to binary problems, this work seeks to address the following question. *If given the same computation time, will scatter search perform better using path relinking or uniform crossover?*

2.3 Automated Parameter Optimization

When comparing different optimization algorithms, benchmark results should not be biased by minor implementation details or parameter tuning. Although this is often discussed in literature (Beiranvand et al., 2017; Eggensperger et al., 2019), many publications on Ising machines and QUBO solvers do not describe their parameter selection process sufficiently (Samorani et al., 2019; Şeker et al., 2020). Still, parameter settings have shown to be an important factor for metaheuristics (Huang et al., 2019). For this reason, many hyper-parameter frameworks exist for finding good sets of parameters for optimization algorithms e.g. *Autotune* (Koch et al., 2018), *Hyperopt* (Bergstra et al., 2015), *Irace* (López-Ibáñez et al., 2016) and more recently proposed *Optuna* (Akiba et al., 2019). *Optuna* is an open source software which uses Bayesian optimization to suggest a set of parameters that lead to more promising performance of the algorithm. *Optuna* has been designed for better functionality and simplicity compared to other existing methods (Akiba et al., 2019). In this work, *Optuna* has been used to derive sets of parameters that lead to more promising results in the scatter search algorithm. Sets of parameters derived are specific to each solution combination method and problem size (e.g. number of knapsack items). Details are provided in Section 5.

3 PROBLEM DEFINITION

0-1 knapsack problems are naturally represented with binary variables. The knapsack problem consists of n items and a constraint. Each item i is defined by a profit $p_i \in \mathbb{N}$ and weight $w_i \in \mathbb{N}$. The aim is to find a subset of items that lead to maximum profit without exceeding the capacity of the knapsack W. We negate the profits to obtain a minimization problem which is more common to use.

$$\min_{x\in\{0,1\}^n} \quad f(x) = -\sum_{i=1}^{n} p_i x_i \tag{3}$$

$$\text{subject to} \quad \left(\sum_{i=1}^{n} w_i x_i\right) \leq W,\ x_i \in \{0,1\} \tag{4}$$

In the QUBO formulation, slack variables are used to encode the weight of a solution (Lucas, 2014). Instead, we will use a non-quadratic constraint function $g(x)$ and multiply it by a penalty weight α (Eq. (5)). The total energy of said system will then be formed by the total knapsack cost $c(x) := f(x)$ from (3) and the weighted penalty.

$$\min_{x\in\{0,1\}^n} \quad E(x) = c(x) + \alpha \cdot g(x) \tag{5}$$

$$\text{where} \quad g(x) = \max\{\sum_{i=1}^{n} w_i x_i - W, 0\} \tag{6}$$

Eq. (6) has been defined such that $g(x) = 0$ holds for every feasible solution x but $g(x)$ increases according to the degree of infeasibility. Furthermore we choose $\alpha = \sum_{i=1}^{n} p_i$ to ensure that no infeasible solution has a lower cost than a feasible one. This formulation is similar to the Binary Quadratic Problem formulation of the Digital Annealer (Hiroshi et al., 2021) where inequality constraints can be defined without using slack variables. Ising machines are physically bound to a maximum number of variables which is why slack variables are often not an appropriate tool for larger problems (Yonaga et al., 2020). We use this formulation to create a conventional setting and align with mappings that are used in state-of-the-art solvers. While this particular problem does not require a QUBO formulation (Eq. (3) does not contain a quadratic term) we can use the same formulation to compare combination methods for other problem sets. The scatter search algorithm presented in this study is therefore directly applicable to any combinatorial optimization problem formulated as QUBO regardless of the presence or absence of quadratic terms.

4 ALGORITHM

4.1 Scatter Search Framework

The main idea of scatter search is as follows. First, a set of elite solutions, called RefSet, is generated.

Algorithm 1: Scatter Search Iteration Loop.

Input: energy function f, set of elite solutions RefSet, index tuples of previously combined solutions UsedParents, number of children to consider as candidates K, minumum distance for updates MinDist

Output: best found solution x^*

```
1  while stopping criteria not met do
2      if unused parents available then
3          Choose parents P_i, P_j ∈ RefSet, i ≠ j
4          UsedParents.insert((i, j))
5          Children ⟵ Combine(P_i, P_j)
6          Candidates ⟵ select K best solutions from Children
7          Candidates ⟵ Improve(Candidates)
8          for Candidate in Candidates do
9              if f(Candidate) < minf(RefSet) or
10             (Distance(Candidate, RefSet) ≥ MinDist and f(Candidate) < max f(RefSet)) then
11                 Replace worst elite solution x_k with Candidate
12                 Remove all entries containing k from UsedParents
13                 x* ⟵ argmin f(RefSet)
14     else
15         RefSet ⟵ Flush(RefSet)
```

To that end, a larger initial population is initialized. Then, an update method is called to populate the reference set with elements from the initial population. From then on, the reference set is maintained and updated with newly generated solutions in an iterative manner. Figure 1 illustrates the entire scatter search, including the initial population step. Algorithm 1 shows the iteration loop, without the initial population step. We will give a more detailed description of our implementation for each method. Parameters are written in italic and discussed in Section 5.

4.2 Implementation Details

4.2.1 Initial Population

A population of P random binary vectors is created. An improvement method (Section 4.2.4) is applied to all of them. Then, the reference set update method is invoked with every item in the population. The process is repeated until the reference set contains *RefSize* items.

4.2.2 Subset Generation

kParents pairs of parents are chosen at random (Line 3 of Algorithm 1). It is made sure that previously chosen pairs are not selected again. If there are no pairs left to choose we *flush the reference set* as described in Section 4.2.7 (Line 15 of Algorithm 1).

4.2.3 Solution Combination

Given multiple pairs of parents, a combination method (see below) is applied once to each pair to generate children (Line 5 of Algorithm 1). The improvement method is only used on the best *ConsiderK* children; the rest is neglected. Afterwards the reference set update is called with every improved solution.

Path Relinking. Starting from P_i, a restricted local search is performed to find a direct path to P_j. Moves are selected in a greedy manner and only variables in which P_i differs from P_j are allowed to be flipped. We make sure no bit is flipped more than once. This way, we generate a set of solutions along a path from P_i to P_j. The length of that path is equal to the Hamming distance between P_i and P_j. The same procedure is repeated to find a path from P_j to P_i. We ignore solutions which only differ from either parent in less than *PathMinDist* bits and return the set of solutions along both paths.

Uniform Crossover. Given two parents P_i, P_j, a new solution x is generated by choosing each variable randomly from each parent.

$$x_k \in_R \{P_j[k], P_j[k]\},\ k = 1, ..., n \tag{7}$$

In practice, we only need to consider variables in which P_i and P_j differ and since x is a binary vector we can simply randomize these entries. This combination method involves less computation steps than

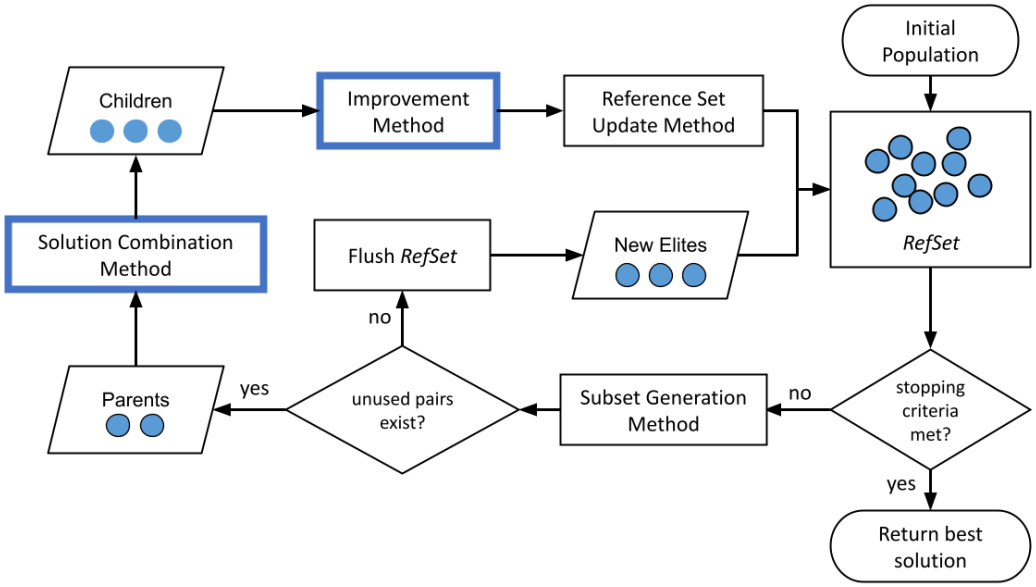

Figure 1: A flow chart of scatter search. Blue points represent solutions. Blue borders indicate methods that can be executed with multiple inputs in parallel. The number of displayed solutions are chosen for illustration purposes only and vary in practice.

path relinking but only produces one child. A child generated by uniform crossover can be thought of as a random item chosen from all solutions on shortest paths between the parents.

4.2.4 Improvement Method

We employ Tabu Search as improvement method (Line 7 of Algorithm 1). It evaluates a 1-flip neighborhood and then chooses the next move in a greedy manner. The last carried out moves are forbidden for the next *TabuTenure* number of iterations. Tabu Search returns the best soltution found after *IterationCutoff* moves without any improvement.

4.2.5 Reference Set Update

A candidate is compared against elite solutions from the reference set. It is accepted if it has a Hamming distance of at least *MinDist* from every elite solution and its cost is lower than the worst item in the reference set (Lines 10 and 10 of Algorithm 1). Said item will be replaced in case of a successful update. An aspiration criterion allows an update if the distance condition is not met but the candidate's cost is lower than the best known solution so far.

4.2.6 Multiprocessing

Some of the above-mentioned tasks are independent of each other and can be performed in parallel. When appropriate we use Python's `multiprocessing` package to

- generate random solutions for initial population,
- improve a set of candidates,
- combine different pairs of parents.

There is a maximum of *MultiMax* worker processes in a pool at any given time. Jobs are completed in an asynchronous manner and subsequent actions await all results before they continue.

4.2.7 Flushing the Reference Set

It is possible that every combination of elite solutions has been combined without any successful update to the reference set. This is not a stopping criterion for our algorithm. Instead, we flush the reference set (Line 15 of Algorithm 1). The best known elite solution is maintained while the others are replaced with newly generated and improved solutions. The process is very similar to how the initial population is built.

4.2.8 Stopping Criteria

We return the best known solution after a fixed time limit or if a given solution quality is met (i.e. a target energy value is reached).

5 EXPERIMENTAL PROTOCOL

The experiments were performed on Ubuntu 20.04.4 LTS with Python 3.7.11 and an AMD Ryzen 9 3950X CPU.

Table 1: List of Parameters.

Name	Description
P	initial population size
RefSize	reference set size
TabuTenure (Init)	number of iterations a move is tabu for initial population
IterationCutoff (Init)	maximum number of TS iterations without update for initial population
MinDist	minimum distance of candidate to reference set
kParents	maximum combinations in each iteration
PathMinDist	minimum distance from either parent to be considered as candidate
ConsiderK	number of best children chosen as candidates
TabuTenure (Comb)	number of iterations a move is tabu for candidate improvement
IterationCutoff (Comb)	maximum number of TS iterations without update for candidate improvement
MultiMax	maximum number of worker processes in a pool

5.1 Parameters

Table 1 lists all parameters. We use *Optuna* (Akiba et al., 2019) to find good parameter settings for each combination method. *Optuna* iteratively samples configurations from a given parameter space and evaluates them. Using this data, it estimates promising values for each parameter and decides which configuration to try out next. We set a fixed time limit of 6 hours for parameter optimization. Performance is evaluated on a subset of two instances with one run per instance for practical reasons. The time limit for each instance was 5 minutes. Each trial with a new configuration should not take too long on its own, otherwise *Optuna* will not be able to sufficiently explore the parameter space. A shorter optimization period would make it more likely to distort the data through very good/bad configurations found by chance. We tried to minimize this effect through a long optimization phase within practical limits. Two sets of parameters are used for Tabu Search. One for the initial population step and another one when it is used as improvement method for new candidates. This way, *Optuna* will be able to balance the intensity of each improvement phase.

5.2 Datasets

The knapsack instances were taken from (Pisinger, 2005). There are different groups of 1-dimensional knapsack instances. We use weakly correlated instances with an item count of $n = 50, 100, 200, 500$. Despite its name, this type of knapsack problem has a very strong correlation between profit and weight of an item which makes it realistic. The return of an investment is proportional to the sum invested within some small variations in most real world applications (Pisinger, 2005).

5.3 Benchmarks

The aim of this work is not to be competitive with other state-of-the-art implementations but to make a comparison of combination methods for the scatter search algorithm. Python alone is reason enough to expect better performance from other algorithms implemented in faster languages like C++. We refer to (Wang et al., 2012) for an extensive comparison of path relinking with several state-of-the-art alternatives on random QUBO and MaxCut problems where it was found to be very effective. For this reason, we only consider *relative* performance in this work.

The experiments were conducted as follows. Parameter optimization was done separately for each problem size. The best configuration found was then used to measure performance. 5 problem instances per size were each tackled 5 times. Runs were initialized with different random seeds. The time limit per run was 5 minutes.

6 RESULTS AND DISCUSSION

Benchmark results are presented in Table 2. Instance filenames are listed alongside their number of knapsack items and the combination method in the first three columns. Columns ‘mean_time’ and ‘std_time’ respectively show mean and standard deviation of the time to reach an optimal solution. We also consider the percentage of the optimal objective value found within the time limit. Columns ‘mean_pct’ and ‘std_pct’ respectively show its mean and standard deviation. We additionally report the average time to reach 100%, 99%, 95% and 90% of the optimal objective in the last four columns. These averages only consider runs which reached the target. Although a time limit of 5 minutes was set for each individual run of the algorithm, it was possible to exceed this limit

by up to 2 seconds due to implementation limitations.

Uniform crossover repeatedly exhibits extremely poor performance on two out of five of the smallest instances. Figure 3 shows that this is consistent across multiple runs on the same problem. Path relinking on the other hand shows less variance and outperforms uniform crossover on all but one instance with 50 knapsack items. It should be noted that uniform crossover arrives considerably faster at suboptimal solutions (90%, 95%) but fails to maintain its convergence rate. The same can be observed for larger instances. Figure 2 shows that path relinking scales better with problem size. The difference is already apparent for 200 knapsack items but is more significant for 500 items. Solution quality for path relinking on these problems is more than 2.5% higher on average at the end of our 5-minute time limit.

Regardless of the combination method, problem instances vary significantly in their difficulty. Performance data on files `knapPI_2_100_1000_2` and `knapPI_2_100_1000_4` illustrates this point very well. The first problem was solved by both algorithms in less than 4 seconds on average while the latter was not solved within 5 minutes in any of the 10 runs. There also seem to be a few instances which are especially unfavorable for a particular method but there was only one such example (`knapPI_2_100_1000_1`) for path relinking in our dataset.

Interestingly, a reference set with only two solutions is found to be ideal for path relinking. Together with the flush functionality described in section 4.2.7 it maintains the best known solution and explores the solution space along paths to random new points. This is true independent of the problem size. *Optuna* always found a reference set of size two to be the best. Uniform crossover on the other hand tends to work best with small reference sets between two and five solutions and does not utilize local search as much. Instead, it rapidly combines the given set of elite solutions in order to improve them. This is feasible for small knapsack problems where the solutions space is not as large and can be explored in this fashion within a reasonable amount of time. However, it does not scale well with larger solution spaces.

Our results suggest that path relinking can explore the solution space more efficiently and is able to reach better solutions within the same time limit. This is despite its higher cost per combination step. Uniform crossover converges fast and is able to quickly achieve mediocre solutions but fails to strategically combine found solutions to effectively promote exploration.

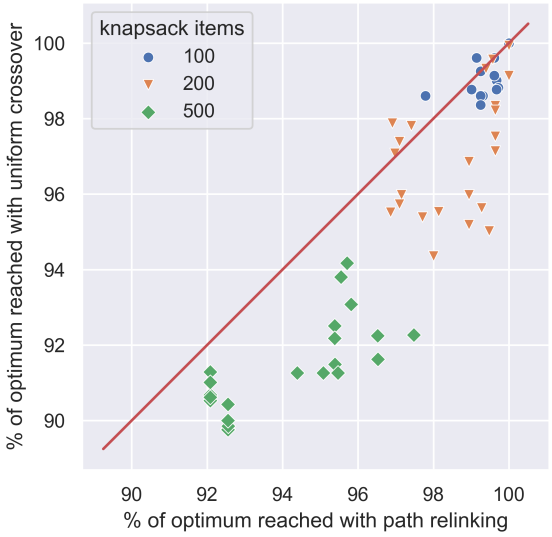

Figure 2: Percentage of optimum energy reached within a time limit of 5 mins. Each point refers to one run. Points in the lower right of the plot correspond to a better performance of path relinking over uniform crossover.

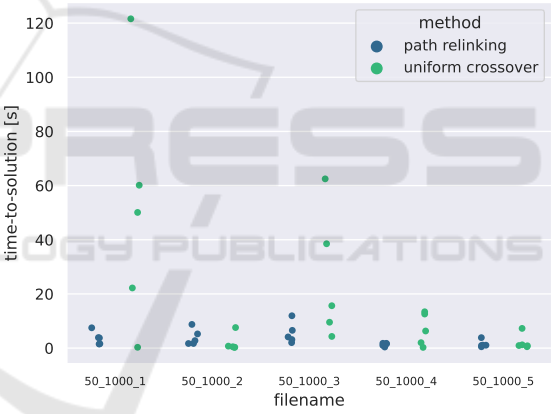

Figure 3: Time-to-solution for knapsack instances with 50 items. Each point corresponds to one run. Filenames are truncated for readability.

7 CONCLUSION AND FURTHER WORK

This work presents a scatter search algorithm for solving QUBO formulations of optimization problems. It features a method of combining local optimal solutions to generate new offspring solutions. We compared two such methods, namely path relinking and uniform crossover. The approach of path relinking was shown to outperform uniform crossover in this setting when applied to the 1-dimensional knapsack problem. Our results suggest that this will also be the case for other optimization problems where uniform

Table 2: Comparing path relinking and uniform crossover in scatter search algorithm: mean and standard deviation percentage optimal cost, mean and standard deviation execution time and mean time to reach 100%, 99%, 95% and 90% of the optimal cost within 5 minutes stopping criteria. Combination method with superior performance for each problem instance is highlighted in bold.

filename	method	size	mean_pct	std_pct	mean_time	std_time	tt_100	tt_99	tt_95	tt_90
knapPI_2_50_1000_1	**path relinking**	50	100.00	0.00	**3.65**	2.43	3.63	2.42	0.99	0.62
	uniform crossover	50	100.00	0.00	50.86	46.01	50.84	28.38	0.63	0.37
knapPI_2_50_1000_2	**path relinking**	50	100.00	0.00	**4.03**	3.01	4.01	4.01	1.29	0.72
	uniform crossover	50	100.00	0.00	1.88	3.20	1.86	1.86	0.35	0.35
knapPI_2_50_1000_3	**path relinking**	50	100.00	0.00	**5.56**	3.94	5.46	5.46	0.74	0.47
	uniform crossover	50	100.00	0.00	26.10	24.17	26.08	26.08	0.29	0.22
knapPI_2_50_1000_4	**path relinking**	50	100.00	0.00	**1.30**	0.59	1.27	1.13	0.54	0.43
	uniform crossover	50	100.00	0.00	6.91	5.98	6.89	0.30	0.23	0.23
knapPI_2_50_1000_5	**path relinking**	50	100.00	0.00	**1.49**	1.34	1.48	1.48	0.70	0.49
	uniform crossover	50	100.00	0.00	2.16	2.87	2.13	2.13	0.64	0.31
knapPI_2_100_1000_1	path relinking	100	100.00	0.00	27.06	28.18	27.04	8.47	3.42	3.42
	uniform crossover	100	100.00	0.00	**5.99**	7.11	5.99	4.88	1.45	1.45
knapPI_2_100_1000_2	path relinking	100	100.00	0.00	3.33	0.58	3.32	3.32	3.32	3.32
	uniform crossover	100	100.00	0.00	**1.68**	0.15	1.67	1.67	1.67	1.67
knapPI_2_100_1000_3	**path relinking**	100	**99.61**	0.36	251.10	109.59	55.05	120.14	3.51	3.51
	uniform crossover	100	99.07	0.53	259.64	90.63	97.50	111.26	1.76	1.76
knapPI_2_100_1000_4	path relinking	100	99.52	0.21	300.11	0.01	-	67.63	2.65	2.65
	uniform crossover	100	99.52	0.21	300.13	0.06	-	178.17	1.64	1.64
knapPI_2_100_1000_5	**path relinking**	100	**98.97**	0.66	300.08	0.05	-	136.54	5.17	3.37
	uniform crossover	100	98.68	0.33	300.16	0.14	-	217.38	7.56	1.50
knapPI_2_200_1000_1	**path relinking**	200	**99.79**	0.29	266.28	40.36	243.69	132.43	30.51	4.71
	uniform crossover	200	99.58	0.36	300.25	0.14	-	157.02	44.03	44.03
knapPI_2_200_1000_2	**path relinking**	200	**99.64**	0.00	300.19	0.13	-	59.64	6.45	6.45
	uniform crossover	200	97.75	0.51	300.19	0.17	-	-	57.74	48.19
knapPI_2_200_1000_3	path relinking	200	97.11	0.19	300.22	0.28	-	-	37.12	5.94
	uniform crossover	200	**97.23**	0.77	300.21	0.06	-	-	71.26	62.16
knapPI_2_200_1000_4	**path relinking**	200	**99.12**	0.25	300.13	0.07	-	211.29	98.64	6.52
	uniform crossover	200	95.74	0.73	300.15	0.13	-	-	155.22	56.36
knapPI_2_200_1000_5	**path relinking**	200	**97.56**	0.56	300.14	0.11	-	-	153.32	118.42
	uniform crossover	200	95.31	0.54	300.18	0.13	-	-	211.04	44.25
knapPI_2_500_1000_1	**path relinking**	500	**96.22**	0.80	300.78	0.41	-	-	100.99	100.91
	uniform crossover	500	93.11	0.87	300.12	0.04	-	-	-	26.65
knapPI_2_500_1000_2	**path relinking**	500	**95.37**	0.77	300.15	0.09	-	-	271.37	183.04
	uniform crossover	500	91.38	0.17	300.18	0.09	-	-	-	27.78
knapPI_2_500_1000_3	**path relinking**	500	**95.38**	0.00	301.12	1.41	-	-	262.32	31.98
	uniform crossover	500	92.25	0.15	300.15	0.06	-	-	-	24.51
knapPI_2_500_1000_4	**path relinking**	500	**92.55**	0.00	301.83	1.61	-	-	-	253.97
	uniform crossover	500	89.96	0.28	300.22	0.08	-	-	-	123.35
knapPI_2_500_1000_5	**path relinking**	500	**92.08**	0.00	300.10	0.05	-	-	-	259.19
	uniform crossover	500	90.83	0.32	300.11	0.05	-	-	-	126.63

crossover is unlikely to scale well with the number of variables.

There are multiple topics for future consideration. First of all, this comparison can be made with different problem formulations. The penalty coefficient α from our energy function in equation (6) is a vibrant topic in research (Diez García et al., 2022; Verma and Lewis, 2020). It would be interesting to see how the two combination methods compare with better adjusted penalty coefficients. The effectiveness of the combination methods for other optimization problems would also be a topic of interest. The Tabu Search could be expanded to employ k-flip moves (Chen, 2015) and make the proposed algorithms more applicable to permutation problems. QUBO formulations where the penalty term $g(x)$ defined in equation (5) is quadratic but no slack variables are needed (e.g. TSP formulation described in (Lucas, 2014)) could use the same energy function but utilize the fast flip-cost evaluation from (2). Other variations of scatter search could also result in better performance. Since uniform crossover is just one popular combination method among many, other methods such as n-point crossover or reduced surrogate crossover could be considered in the same way. Parameter optimization could also be greatly improved. Trial times present a challenge and it is unclear how a different number of solver runs per trial would have affected our results. Pruning and racing techniques could be applied to stop less promising trials early and achieve better performance (Birattari, 2009).

REFERENCES

Akiba, T., Sano, S., Yanase, T., Ohta, T., and Koyama, M. (2019). Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '19, page 2623–2631, New York, NY, USA. Association for Computing Machinery.

Alkhamis, T. M., Hasan, M., and Ahmed, M. A. (1998). Simulated annealing for the unconstrained quadratic pseudo-boolean function. *European Journal of Operational Research*, 108(3):641–652.

Amini, M. M., Alidaee, B., and Kochenberger, G. A. (1999). *A Scatter Search Approach to Unconstrained Quadratic Binary Programs*, page 317–330. McGraw-Hill Ltd., UK, GBR.

Aramon, M., Rosenberg, G., Valiante, E., Miyazawa, T., Tamura, H., and Katzgraber, H. G. (2019). Physics-inspired optimization for quadratic unconstrained problems using a digital annealer. *Frontiers in Physics*, 7:48.

Ayodele, M. (2022). Comparing the digital annealer with classical evolutionary algorithm.

Beiranvand, V., Hare, W., and Lucet, Y. (2017). Best practices for comparing optimization algorithms. *Optimization and Engineering*, 18(4):815–848.

Bergstra, J., Komer, B., Eliasmith, C., Yamins, D., and Cox, D. D. (2015). Hyperopt: a python library for model selection and hyperparameter optimization. *Computational Science & Discovery*, 8(1):014008.

Birattari, M. (2009). *Tuning Metaheuristics: A Machine Learning Perspective*. Springer Publishing Company, Incorporated, Berlin, Heidelberg, 1st ed. 2005. 2nd printing edition.

Cao, Z., Zhang, Y., Guan, J., and Zhou, S. (2018). Link prediction based on quantum-inspired ant colony optimization. *Scientific reports*, 8(1):1–11.

Chen, W. (2015). Optimality conditions for the minimization of quadratic 0-1 problems. *SIAM Journal on Optimization*, 25(3):1717–1731.

Cipra, B. A. (1987). An introduction to the ising model. *The American Mathematical Monthly*, 94(10):937–959.

Deng, W., Liu, H., Xu, J., Zhao, H., and Song, Y. (2020). An improved quantum-inspired differential evolution algorithm for deep belief network. *IEEE Transactions on Instrumentation and Measurement*, 69(10):7319–7327.

Diez García, M., Ayodele, M., and Moraglio, A. (2022). Exact and sequential penalty weights in quadratic unconstrained binary optimisation with a digital annealer. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '22, New York, NY, USA. Association for Computing Machinery.

Dunning, I., Gupta, S., and Silberholz, J. (2018). What works best when? a systematic evaluation of heuristics for max-cut and qubo. *INFORMS Journal on Computing*, 30(3):608–624.

Eggensperger, K., Lindauer, M., and Hutter, F. (2019). Pitfalls and best practices in algorithm configuration. *Journal of Artificial Intelligence Research*, 64:861–893.

Şeker, O., Tanoumand, N., and Bodur, M. (2020). Digital annealer for quadratic unconstrained binary optimization: a comparative performance analysis.

Fujimoto, N. and Nanai, K. (2021). Solving qubo with gpu parallel mopso. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, GECCO '21, page 1788–1794, New York, NY, USA. Association for Computing Machinery.

Glover, F. and Hao, J.-K. (2010). Efficient evaluations for solving large 0-1 unconstrained quadratic optimisation problems. *International Journal of Metaheuristics*, 1(1):3–10.

Glover, F., Kochenberger, G. A., and Alidaee, B. (1998). Adaptive memory tabu search for binary quadratic programs. *Management Science*, 44(3):336–345.

Hakli, H. and Ortacay, Z. (2019). An improved scatter search algorithm for the uncapacitated facility location problem. *Computers & Industrial Engineering*, 135:855–867.

Hiroshi, N., Junpei, K., Noboru, Y., and Toshiyuki, M. (2021). Third generation digital annealer technology.

Huang, C., Li, Y., and Yao, X. (2019). A survey of automatic parameter tuning methods for metaheuristics. *IEEE transactions on evolutionary computation*, 24(2):201–216.

Johnson, M. W., Amin, M. H., Gildert, S., Lanting, T., Hamze, F., Dickson, N., Harris, R., Berkley, A. J., Johansson, J., Bunyk, P., et al. (2011). Quantum annealing with manufactured spins. *Nature*, 473(7346):194–198.

Koch, P., Golovidov, O., Gardner, S., Wujek, B., Griffin, J., and Xu, Y. (2018). Autotune: A derivative-free optimization framework for hyperparameter tuning. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, KDD '18, page 443–452, New York, NY, USA. Association for Computing Machinery.

Kochenberger, G., Hao, J.-K., Glover, F., Lewis, M., Lü, Z., Wang, H., and Wang, Y. (2014). The unconstrained binary quadratic programming problem: a survey. *Journal of combinatorial optimization*, 28(1):58–81.

López-Ibáñez, M., Dubois-Lacoste, J., Cáceres, L. P., Birattari, M., and Stützle, T. (2016). The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58.

Lucas, A. (2014). Ising formulations of many NP problems. *Frontiers in Physics*, 2:Article 5.

Matsubara, S., Takatsu, M., Miyazawa, T., Shibasaki, T., Watanabe, Y., Takemoto, K., and Tamura, H. (2020). Digital annealer for high-speed solving of combinatorial optimization problems and its applications. In *2020 25th Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 667–672, Beijing, China. IEEE.

Merz, P. and Katayama, K. (2004). Memetic algorithms for the unconstrained binary quadratic programming problem. *BioSystems*, 78(1-3):99–118.

Moreno, E., Espinoza, D., and Goycoolea, M. (2010). Large-scale multi-period precedence constrained

knapsack problem: a mining application. *Electronic notes in discrete mathematics*, 36:407–414.

Naseri, G. and Koffas, M. A. (2020). Application of combinatorial optimization strategies in synthetic biology. *Nature communications*, 11(1):1–14.

Palubeckis, G. (2004). Multistart tabu search strategies for the unconstrained binary quadratic optimization problem. *Annals of Operations Research*, 131:259–282.

Picek, S., Golub, M., and Jakobovic, D. (2011). Evaluation of crossover operator performance in genetic algorithms with binary representation. In *Proceedings of the 7th International Conference on Intelligent Computing: Bio-Inspired Computing and Applications*, ICIC'11, page 223–230, Berlin, Heidelberg. Springer-Verlag.

Pisinger, D. (2005). Where are the hard knapsack problems? *Comput. Oper. Res.*, 32(9):2271–2284.

Samorani, M., Wang, Y., Lv, Z., and Glover, F. (2019). Clustering-driven evolutionary algorithms: an application of path relinking to the quadratic unconstrained binary optimization problem. *Journal of Heuristics*, 25(4):629–642.

Soloviev, V. P., Bielza, C., and Larranaga, P. (2021). Quantum-inspired estimation of distribution algorithm to solve the travelling salesman problem. In *2021 IEEE Congress on Evolutionary Computation (CEC)*, pages 416–425, Kraków, Poland. IEEE.

Sorensen, K., Sevaux, M., and Glover, F. (2017). A history of metaheuristics.

Supasil, J., Pathumsoot, P., and Suwanna, S. (2021). Simulation of implementable quantum-assisted genetic algorithm. In *Journal of Physics: Conference Series*, volume 1719, page Article 012102, Trang, Thailand. IOP Publishing.

Toth, P. and Vigo, D., editors (2001). *The Vehicle Routing Problem*. Society for Industrial and Applied Mathematics, USA.

Umbarkar, A. J. and Sheth, P. D. (2015). Crossover operators in genetic algorithms: a review. *ICTACT journal on soft computing*, 6:Article 1.

Verma, A. and Lewis, M. (2020). Penalty and partitioning techniques to improve performance of qubo solvers. *Discrete Optimization*, 44:100594.

Wang, Y., Lü, Z., Glover, F., and Hao, J.-K. (2012). Path relinking for unconstrained binary quadratic programming. *European Journal of Operational Research*, 223:595–604.

Yasudo, R., Nakano, K., Ito, Y., Tatekawa, M., Katsuki, R., Yazane, T., and Inaba, Y. (2020). Adaptive bulk search: Solving quadratic unconstrained binary optimization problems on multiple gpus. In *49th International Conference on Parallel Processing - ICPP*, ICPP '20, New York, NY, USA. Association for Computing Machinery.

Yonaga, K., Miyama, M. J., and Ohzeki, M. (2020). Solving inequality-constrained binary optimization problems on quantum annealer.

Zainuddin, F. A., Abd Samad, M. F., and Tunggal, D. (2020). A review of crossover methods and problem representation of genetic algorithm in recent engineering applications. *International Journal of Advanced Science and Technology*, 29(6s):759–769.