UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

TADEU ZUBARAN

Multi-point search for combinatorial optimization problems

Final Report presented in partial fulfilment
of the requirements for the degree of
Bachelor in Computer Science

Prof. Dr. Marcus Ritt
Advisor

Porto Alegre, march 2013

CIP – CATALOGING-IN-PUBLICATION

Zubaran, Tadeu

Multi-point search for combinatorial optimization problems / Tadeu Zubaran. – Porto Alegre: PPGC da UFRGS, 2013.

54 f.: il.

Final Report (Master) – Universidade Federal do Rio Grande do Sul. BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO, Porto Alegre, BR–RS, 2013. Advisor: Marcus Ritt.

1. Combinatorial optimization. 2. Heuristics. 3. UFRGS. 4. Go with the winners. I. Ritt, Marcus. II. Título.

“Beatings will continue until morale improves.”
— Unknown

AGRADECIMENTOS

To my mother.

CONTENTS

LIST OF ABBREVIATIONS AND ACRONYMS

GRASP	Greedy Randomized Adaptive Search Procedure
GWTW	Go With The Winners
IGWTW	Iterative Go With The Winners
MR	Machine Reassignment
SA	Simulated Annealing
SR	Simple Restart
TSP	Travelling Salesperson Problem

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

In this work we propose and test ways to apply the principles of go with the winners in combinatorial optimization problems. Go with the winners was proposed as an algorithm to move particles in abstract trees, and is compared with an algorithm, the *simple restart*, where several particles are released at the root of the tree and let to descend the tree without interacting with one another. The go with the winners approach lets particles use information from other particles in order to increase the chance that they will reach deeper levels of the tree.

We develop an algorithm that use the core ideas of the go with the winners in three distinct problems. First we define an artificial optimization problem to test a preliminary algorithm and see how the theoretical predictions are translated in a mid point between an actual optimization problem and the abstract framework. We proceed, then, to implement and test the go with the winners in a classic optimization problem, the travelling salesperson problem, and finally in the modern problem machine reassignment.

For each of our implementations we perform a comprehensive set of benchmark tests and compare the performance against the simple restart and the commonly used metaheuristics GRASP and simulated annealing.

Keywords: Combinatorial optimization, heuristics, UFRGS, go with the winners.

Busca multi-ponto para problemas de otimizaçõ combinatória.

RESUMO

Neste trabalho propomos e testamos maneiras de aplicar os princípios por traz do go with the winners, em problemas de otimização combinatória. Go with the winners foi proposto como um algorítimo para mover partículas em árvores abstratas, e é comprado com um algorítimo, simple restart, onde diversas partículas são soltas na raiz da árvore e deixadas à descender sem interagirem umas com as outras. O técnica go with the winners permite que as partículas utilizem informação das outras de forma a aumentar a chance delas chegarem em níveis mais baixos da árvore.

Nós desenvolvemos um algorítimo que utiliza as idéias chave do go with the winners para três problemas distintos. Primeiramente desenvolvemos um problema artificial para testes preliminares do algorítimo para verificarmos como as previsões teóricas são traduzidas para um ponto intermediário entre um verdadeiro problema de otimização e o framework abstrato. Nós então implementamos e testamos o go with the winners no clássico problema de otimização do caixeiro viajante e por último no moderno *machine reassignment*.

Para cada uma de nossas implementações nós fazemos uma bateria completa de testes e comparamos com a performance contra o *simple restart* e as meta-heurísticas comumente utilizadas GRASP e *simulated annealing*.

Palavras-chave: otimização combinatória, heurísticas, UFRGS, go with the winners.

1 GO WITH THE WINNERS

1.1 Introduction

Many of the interesting instances of NP-Hard problems are not solvable exactly in a timely manner with today's technology and, if $P \neq NP$, it is unlikely that hardware advances alone will solve the problem. We can use heuristic algorithms to get good approximations if the situation does not require a guaranteed optimal solution.

Performing local search means we will explore the vicinity of a given state always going towards better solutions. if the objective function is not convex local search is not guaranteed to find the best solution, because it will likely get stuck in local minima. Several meta-heuristics use local searches but employ techniques to surmount these local minima in the hope of eventually finding the best global solution.

We can form different set of rules that dictate how our local search will behave. If our local search is not deterministic we can run the algorithm several times and eventually find the best solution. There is waste in this approach as we throw away information from previous runs.

We can think of a local search as a set of rules to move a particle in a state space, so we can, instead of releasing only one particle at a time, release several particles simultaneously and from time to time classify them as doing badly or doing well. If we move the particles that are doing badly to the states of the particles that are doing well we can hope to find better results than if we just let them evolve isolated. This is the idea behind the go with the winners approach, proposed in (Aldous and Vazirani 1994). There is a similarity between this approach and genetic algorithms, since we are incorporating a survival of the fittest type rule without a mating rule.

1.2 Finding Deep Vertices On a Tree

To have a rigorous analysis of the go with the winners scheme (henceforth called GWTW) we model the randomized optimization algorithm as a set of rules to move a particle in a state space and the state space as a rooted tree in which the particle moves from the root to a leaf. This models the search space of a typical local search, which only allows solutions that improve the current solution. Thus the initial solution is a tree of always better solutions with no cycles (although two paths can be joined later, we can conceptually treat them as different solutions).

In our scheme Algorithm 1 models the behaviour of a randomized local search. In harder problems the local search has typically low probability of reaching the best solution. This will be translated in deep and "unbalanced" trees were a particle moving randomly following the probabilities of transitions in the tree will have low chances of

reaching the deepest nodes.

We introduce the simple restart algorithm (SR) defined in Algorithm 2. Is is introduced in Aldous and Vazirani (1994) (where it is called "algorithm 0") and will serve as a baseline for experimental comparison in later chapters. The algorithm consists on consecutive independent runs of a local search, keeping track of the best depth achieved by those local searches.

Algorithm 3 models the go with the winners approach, where we release multiple particles in random walks, each one isolated behaving as Algorithm 2. They interact in the sense that once a particle reaches a dead end it goes to a state where there is still chance of improvement.

We have a search tree, with an associated transition probability for all edges. We call the set of all particles *beam*, the parameter B determines how many particles the beam contains. All particles start at the root of the tree, and at each step we move each particle to one of its children nodes. The particles will, eventually, follow distinct paths because the transitions are not deterministic. Whenever a particle reaches a leaf we put it in the same place as a random particle that still is in an inner node. In case of more than one particle reaching a leaf at the same time we distribute them evenly (no state get two or more particles than any other) and randomly.

This models the search space of a typical indeterministic local search, which allows solutions that improve the current solution, and, although two paths can be joined later, we can conceptually treat them as different solutions.

Algorithm 1 Local Search

while particle is not in leaf node **do**
  Move the particle to a random child of current state.
end while

Algorithm 2 Simple Restart

Start one particle at the root of the tree.
while time not exceeded **do**
  Perform local search.
  Restart particle at the root of the tree.
end while

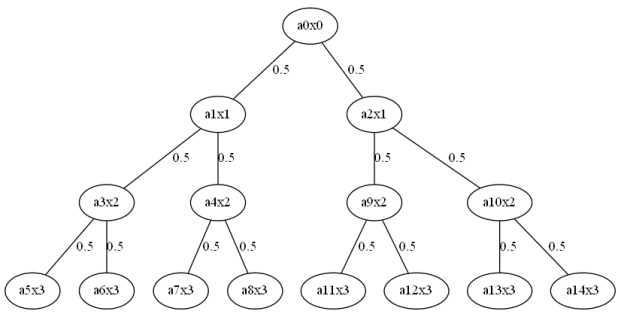
In the model there are two important properties of a tree, the depth and if it is balanced, which in Aldous and Vazirani (1994) is the variable κ. κ provides a quantitative measurement of how unbalanced a trees is. We define κ in equation (1.1a). The set V_i is the set of nodes at depth i (the root has depth 0) and the other variables are probabilities defined using Algorithm 1. The probability $a(i)$ is the chance that the particle reaches at least level i, $p(v)$ that it will visit vertex v and $a(j|v)$ given that the particle visits vertex v the probability that it will reach at least level j.

$$\kappa = \max_{0 \le i < j \le d} \kappa_{i,j} \tag{1.1a}$$

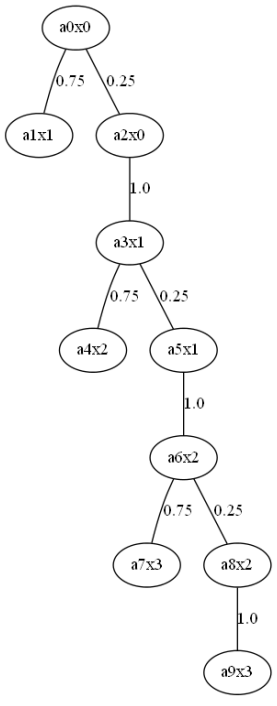
$$\kappa_{i,j} = \frac{a(i)}{a^2(j)} \sum_{v \in V_i} p(v) a^2(j|v) \ge 1 \tag{1.1b}$$

Algorithm 3 Go With The Winners

Start B particles at the root of the tree.
while at least one particle is not on a leaf **do**
  Let L be the set of particles in leafs.
  Let M be the set of particles in non-leafs.
  while $|L| > |M|$ **do**
    Let R a set with $|M|$ random particles from L
    Remove the particles in R from L
    Randomly assign each particle in R to a different one in M
    Move each particle in R to the state of its assigned particle.
  end while
  Randomly assign each remaining particle in L to a different one in M
  Move each particle in L to the state of its assigned particle.
  Move the particle to a random child of current state.
end while



(a) An easy tree. We will eventually reach the deepest level, there is no chance we will get stuck.



(b) A Hard tree. We need luck to reach the deepest level.

Figure 1.1: Two examples of the kind of trees we may find in a problem.

In the example of the trees in Figure 1.1 the tree on the left has $\kappa = 1$ and the one on the right has $\kappa = 4$.

The interesting result in the original GWTW paper, that motivates us to try to evaluate such strategy in practice, is stated in Theorem 1. Algorithm 2 has exponential worst case growth, so , at some point we should have an advantage of this approach in local searches.

Theorem 1 *There is a polynomial poly() such that if Algorithm 3 run with* $B = \kappa \times poly(d)$ *then it fails to find the deepest leaf with probability at most* $1/4$.

In Saha (2013) we have an study on the GWTW approach that shows that too many particles can actually decrease the chance of reaching the deepest vertices. If we choose a big beam with too many particles we will have the costs of managing more particles and also a decrease on our chance of achieving good results, which reinforces the importance of this parameter.

1.3 Objectives

GWTW and SR can be faithfully translated to an heuristic algorithm for optimization problems. We think about the states in the search space as the nodes in the abstract tree. We always improve the solution at each step so it is impossible to search in cycles. The relevant information about a particle is its position in the tree, which is a state in the search space. The movement of the particle in the tree is represented, in the optimization problem, as the transformation of the state into another state according to a chosen neighbourhood. We have no restrictions on the state associated with the root of the tree.

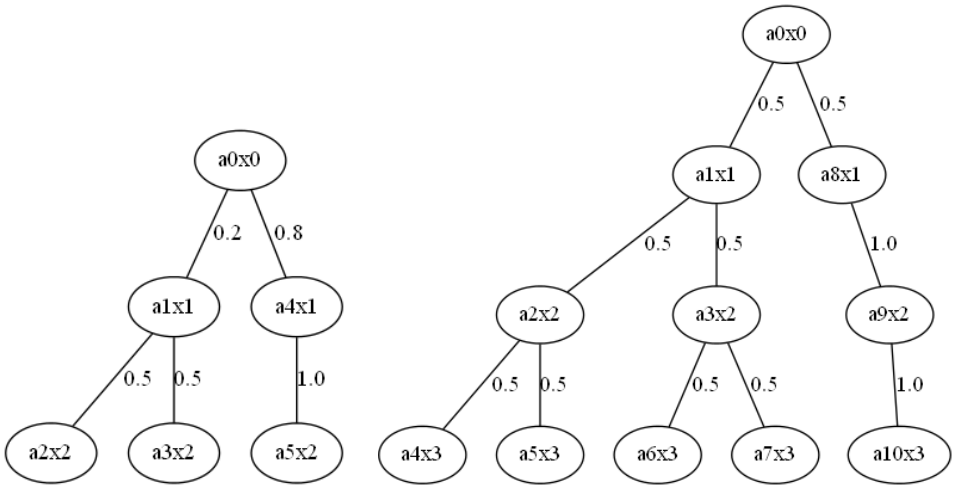
The particles must navigate the tree in a non-deterministic way, this means that we must use a non-deterministic way of computing a neighbour of a state. When we are computing a new state that improves the current one we have two basic options: we may return as soon as we find a state that is better than the current one, so called first improvement, or we can search until we find the best node in the neighbourhood, so called best improvement. It is natural to choose first improvement in our case because, if we search the neighbourhood in a random order the first improvement will be non-deterministic, while best improvement will be, usually, deterministic, i.e. if there is only a single best neighbour.

We are counting hops as if they always improved the solution found by the same amount and as if they cost the same to compute, but this is rarely the case. It is fair to assume that more hops tend to improve the solution more than less hops, but there might be search spaces where this does not hold.

We first implement a preliminary algorithm on artificial instances to draw a baseline expectation and develop insight on the theory. Next we apply the GWTW and SR algorithms in two real optimization problems, along with other traditional meta-heuristics. Our main objective is access if the theoretical predictions translate accordingly to expectation and gauge the viability of GWTW when compared with traditional techniques.

1.4 Artificial Instances

As a preliminary study we perform a set of tests comparing the GWTW to the traditional approaches in artificial examples. We create a family of trees to perform tests in the terms that we are interested in, i.e. quality of solution found by time it takes to find it. Time in this context is represented by the number of moves of the particles. This serves



(a) $t_2^{0.8}$ (b) $t_3^{0.5}$

Figure 1.2: Examples of the basic block of our synthetic family of trees.

not only to give us better understanding of the go with the winners strategy but also to serve as a baseline of what can be expected in the real experiments.

1.4.1 Basic Block t_m^α

In this section we define and study the artificial family of search trees in which we will perform our set of tests.

First we define $t_m^\alpha, (0 < \alpha < 1)$ that will be the building block of the $T_{m,n}^\alpha$ family. t_m^α is a root node which has two children, one child is the root of a binary complete tree of depth m and the other child is the root of a line also with depth m. The transition probability for the binary tree is irrelevant for all our purposes but we define as one half for each child for all nodes, the chance of transition in the line has to be one for all nodes and for t_m^α's root α is the chance of transition to the line sub-tree, therefore the chance to transition to the binary complete sub-tree is $\alpha - 1$. Examples of t_m^α can be seen in Figure 1.2.

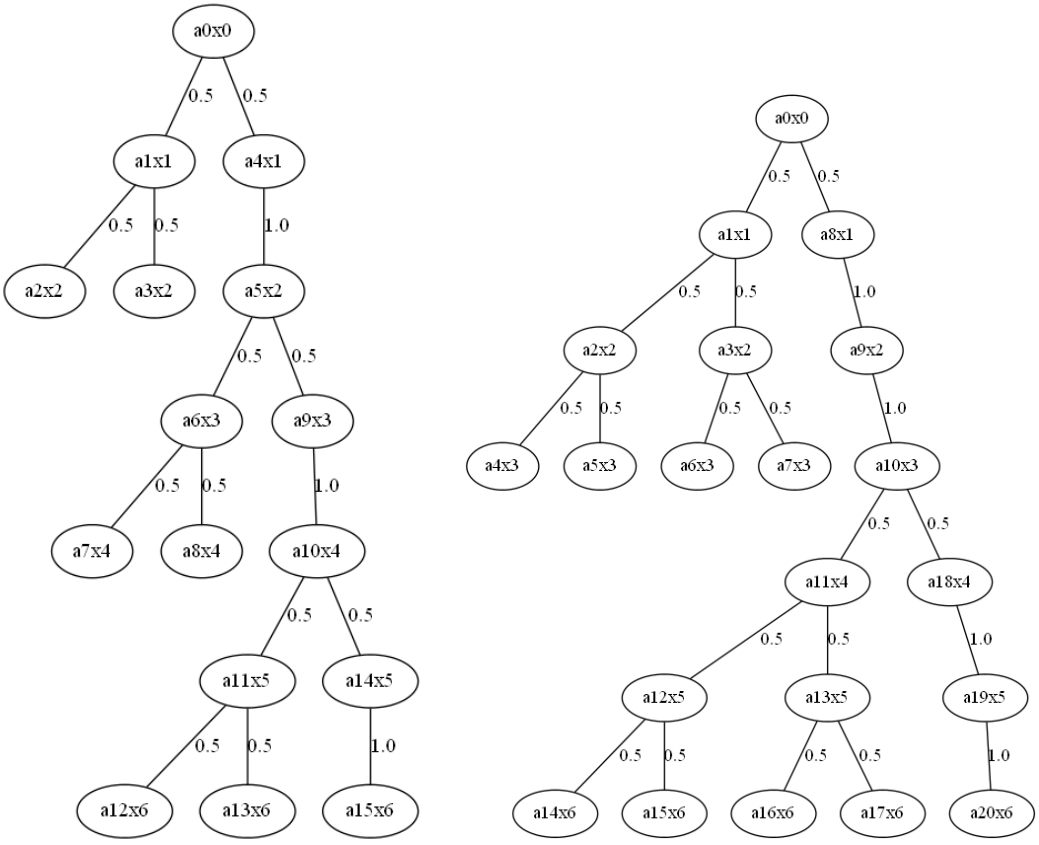
1.4.2 Family $T_{m,n}^\alpha$

$T_{m,n}^\alpha$ $m, n \in \mathbb{N}^*, \alpha \in \mathbb{R} \wedge (0 < \alpha < 1)$ is built by attaching n t_m^α together. The last node in the line sub-tree of each t_m^α becomes the father of the next t_m^α's root. Figure 1.3 contains two example of $T_{m,n}^\alpha$.

This set of trees is inspired by the examples in Aldous and Vazirani (1994), as they use this sort of tree to illustrate their findings, but they don't define the family in the article. It is suitable for our purposes because it gives us a good variety of trees to experiment, and we can modify the difficulty of finding the deepest vertex by changing the depth or the probability of falling in the dead end branch of each t_m^α, as well as get any κ we desire by tuning $T_{m,n}^\alpha$ as shown in Theorem 2:

Theorem 2 *We want to calculate $\kappa(\alpha)$ for any arbitrary $T_{m,n}^\alpha$.*

$$\kappa_{i,j} = \frac{a(i)}{a^2(j)} \sum_{v \in V_i} p(v) a^2(j|v)$$



(a) $T_{2,3}^{0.5}$ (b) $T_{3,2}^{0.5}$

Figure 1.3: Two elements of our family of trees.

If i and j are in the same t_m^α: Let x_u be the root in level u of the t_m^α and $p(x_u) = \eta$

$$\kappa_{u,j} = \frac{(\eta/\alpha)}{\eta^2}(\eta) = 1/\alpha$$

$$\kappa_{i\neq u,j} = \frac{(\eta)}{\eta^2}(\eta) = 1$$

If i and j are not in the same t_m^α, we do not need to consider the root node of any t_m^α but the first because the node also belong to the previous t_m^α. Let ω be the number of t_m^α between the two levels considered and $a(i) = \delta$.

$$\kappa_{i\neq 0,j} = \frac{\delta}{(\delta\alpha^{\omega+1})^2}(\delta\alpha\alpha^{2\omega}) = 1/\alpha$$

$$\kappa_{0,j} = \frac{1}{\alpha^{2(\omega+1)}}(\alpha^{2(\omega+1)}) = 1, \textit{ with j out of the first } t_m^\alpha$$

therefore $\kappa = \max_{0\leq i<j\leq d} \kappa_{i,j} = 1/\alpha$, since $0 < \alpha < 1$.

1.5 Results

Tables 1.1 and 1.2 give us the average depth each algorithm achieves in $T_{8,8}^\alpha$ and $T_{4,4}^\alpha$ respectively. We have GWTW with different number of particles and SR with the same number of particles, so we can have a fair comparison. We also have the results for SR with one particle, which is a randomized local search. We regulate $\alpha = 1/\kappa$ to have the results for different κ for each tree, in this way we can see the performance of the algorithms in trees with variable hight and variable chance of reaching a leaf at each level. The column "%" gives us the mean of how much better, in percent, each of the GWTW is in relation with SR with the same amount of particles as in Equation (1.2).

$$\% = \frac{100}{3} \times \left(\frac{GWTW\ 2}{SR\ 2} + \frac{GWTW\ 4}{SR\ 4} + \frac{GWTW\ 8}{SR\ 8} - 3\right) \tag{1.2}$$

GWTW systematically reaches deeper levels in comparison with SR with the same amount of particles. The difference in performance varies according to κ and the number of particles. We can see in the column "%" the tendency of the SR to have performance closer to GWTW when κ is higher. Figure 1.4 which plots the average depth by κ shows the diminishing advantage.

In an optimization problem κ would be a measurement of the likelihood of reaching local minima search, while the depth of the tree would be a measurement of the size of the search space that local search can explore. We can expect to have punctually better configurations of beam size according to the different search spaces, but we have an indication that GWTW should achieve better solutions in comparison to SR if we properly adjust the beam size. In the concrete problems in next chapters we adjust the number of particles for each problem before performing a comprehensive evaluation of the technique.

In Tables 1.1 and 1.2 as expected we have better results with more particles. More particles have more chances to reach deeper levels but at the same time particles will have

Table 1.1: Average depth by algorithm in $T^{\alpha}_{4,4}$

κ	SR 1	GWTW 2	SR 2	GWTW 4	SR 4	GWTW 8	SR 8	%
2	7.5	10.932	9.732	14.452	12.244	15.84	14.248	14
4	5.408	7.068	6.472	9.98	8.048	13.832	9.524	26
8	4.54	5.184	5.072	6.632	5.948	9.236	7.012	15
16	4.224	4.576	4.512	5.06	4.996	6.456	5.8	4
32	4.144	4.26	4.288	4.46	4.472	5.208	4.94	2

SR x means run Algorithm 2 x times and keep best solution.
GWTW x means run one time Algorithm 3 with $B = x$
Average taken over 1000 runs.

Table 1.2: Average depth by algorithm in $T^{\alpha}_{8,8}$

κ	SR 1	GWTW 2	SR 2	GWTW 4	SR 4	GWTW 8	SR 8	%
2	15.92	28.73	21.28	51.48	27.80	63.04	35.12	66
4	10.50	14.64	12.52	23.99	16.22	45.57	19.75	64
8	9.27	10.12	10.24	13.72	11.91	22.60	14.44	33
16	8.59	9.11	9.144	10.21	9.95	13.46	11.56	6
32	8.32	8.53	8.456	9.04	9.088	10.33	9.79	2

SR x means run Algorithm 2 x times and keep best solution.
GWTW x means run one time Algorithm 3 with $B = x$
Average taken over 1000 runs.

different life cycles in each algorithm. An implementation of the SR will not perform the same number of operations as a GWTW with the same number of particles. In Tables 1.3 and 1.4 we see the number of steps each algorithm took to reach a certain level. GWTW will always perform n movement of particles at each level, where n is the number of particles, since even if one particle reaches a leaf it will be put in an inner node and continue to descent. SR with n particles will always perform, at most, n particle movements at each level, if all articles reach that level, and usually will perform less as shown in both Tables 1.3 and 1.4.

All values analysed are influenced by this particular kind of tree $T^{\alpha}_{m,n}$. We use this study as a guideline but we may encounter topologies for which these results do not hold.

Randomized Local search is a special case of GWTW with one particle. GWTW with more particles will, usually, search a larger part of the search space than a local search, since with more particles the number of states investigated tends to increase, yet the states reachable to GWTW do not increase with the number of particles. We are still searching the same tree as local search, just more thoroughly.

We will compare GWTW with simple restart, but SR is not used in real problems, so we will also compare GWTW with two meta-heuristics GRASP and simulated annealing in next chapters as well. It is important to note that GWTW does not share important properties with this techniques. In our implementation we do not allow a solution to move to a neighbour that has worse objective function value. This means that both GRASP and simulated annealing have, potentially, access to a bigger part of the search space than GWTW. Other distinction is that both meta-heuristics can run for an undetermined amount of time. GWTW have what we call a natural stop condition, when all particles are in leafs, meaning it can't run indefinitely.

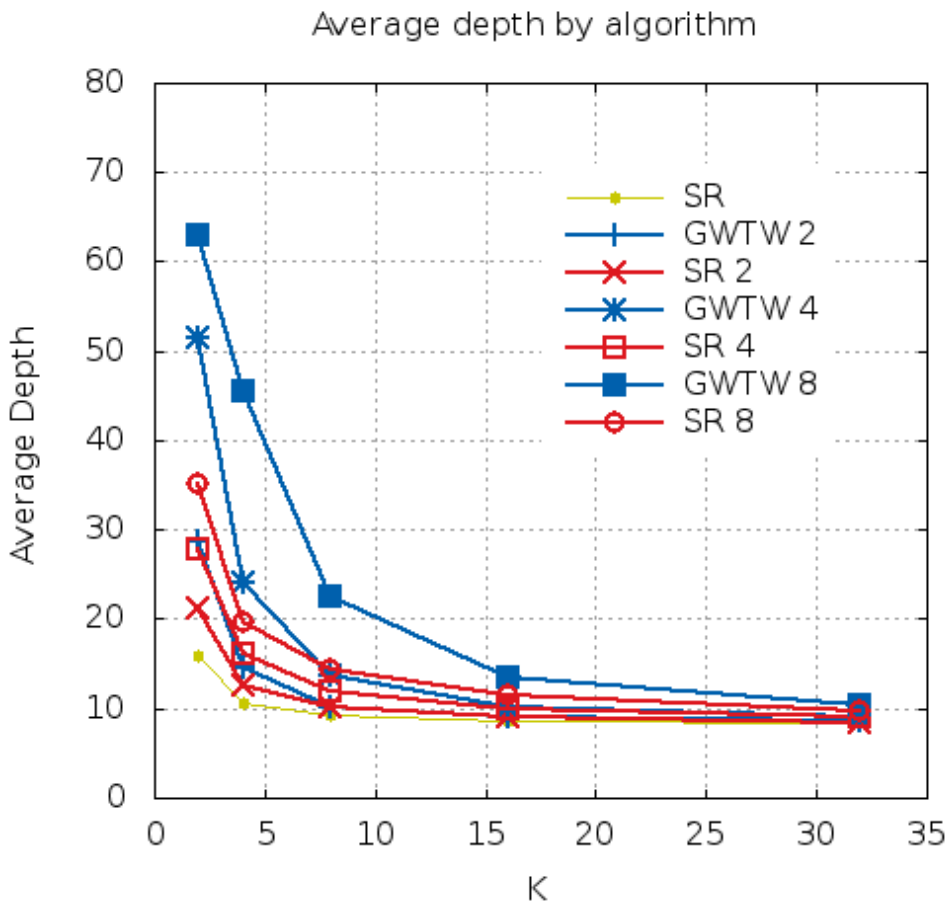
Table 1.3: Average steps taken by level advanced in $T^{\alpha}_{4,4}$

Algorithm	Steps By Level
SR 1	1
SR 2	1.72
GWTW 2	2
SR 4	2.89
GWTW 4	4
SR 8	4.96
GWTW 8	8

Table 1.4: Average steps taken by level advanced in $T^{\alpha}_{8,8}$

Algorithm	Steps By Level
SR 1	1
SR 2	1.70
GWTW 2	2
SR 4	2.81
GWTW 4	4
SR 8	4.63
GWTW 8	8

Figure 1.4: Average depth per algorithm in $T^{\alpha}_{8,8}$ by κ



2 TSP

2.1 Introduction

The travelling sales person problem, henceforth called TSP, is a classic NP-Hard optimization problem (Gutin and Punnen 2002). The input is an undirected graph with weighted edges. A valid solution is a cycle that goes through every node exactly once, known as a Hamiltonian Cycle. The cost associated with each valid solution is the sum of the cost of all edges in it. The objective is to find a valid solution with the lowest cost possible.

TSP is commonly interpreted as the problem of finding the smallest distance one must travel in order to visit every city in some map once. The input graph nodes are interpreted as the cities and the edge weights as the distance between those cities. Despite this common interpretation the definition still makes sense for non-euclidean graphs.

We chose the TSP as our first problem to apply GWTW, because it is commonly used to test new algorithms, it has a comprehensive set of instances for benchmarking and a few more desirable characteristics explained in Section 2.3. We do not expect to find new best solutions for the standard problems, since it is a thoroughly studied problem. The objective of this project is to gauge the effectiveness of the GWTW approach.

2.2 Go With The Winners, Simple Restart and GRASP

The formal framework in Aldous and Vazirani (1994) presented in Section 1 gives us an indication that GWTW has the potential to outperform SR yet do not compare GWTW to GRASP (Feo and Resende 1995).

The meta-heuristic GRASP defined in Algorithm 4 systematically performs local searches. Each time we perform the search we start from a different root node. Each root node is build with a greedy-randomized heuristic. We want good initial solutions from the greedy approach, but we want to start from different starting nodes each time.

The greedy randomized construction is shown in Algorithm 5. We first compute the cost of adding each element to the solution, next we build the restricted candidate list, which depends on α. We select a random element from the list and add it to the current solution. We reevaluate the costs of the remaining elements and repeat until we have a complete solution.

To build the restricted candidate list we include each remaining element that has cost below a threshold, which is given by $\alpha(C_{max} - C_{min})$, where C_{max} and C_{min} are the maximum cost and minimum cost of all the candidate elements.

GRASP is related to the SR, since both perform continuously a restart followed by a local search. The restart process in both is different: We have the randomized greedy

procedure in GRASP where in SR we have a static root. We have good theoretical indications that GWTW can outperform the simple restart algorithm so it is also interesting to compare it to the GRASP meta-heuristic as well.

Algorithm 4 GRASP

```
Parameter α
while time is not up do
  Construct greedy-randomized(α) initial state.
  Perform local search
  Update best solution found.
end while
```

Algorithm 5 Greedy randomized construction

```
Parameter α.
Evaluate the cost of adding each element to the solution.
while solution is not complete do
  Build the restricted candidate list.
  Select an element from the restricted candidate list
  Add element to the solution.
  Reevaluate the cost of adding each remaining element to the solution.
end while
```

2.3 Implementation

First we need to choose a neighbourhood. Given a Hamiltonian Cycle, which is a node in the search space graph, we need to define how to construct its neighbours. We choose a classical neighbourhood, the 2-opt (Croes 1958). To generate a neighbour in the 2-opt neighbourhood we remove any two edges of the path and reconnect the loose vertices the only way possible that will not produce the same path again. In Figure 2.1 we see an illustration of the process.

The 2-opt neighbourhood suits our purposes. It is a considerably big neighbourhood, since every node in the search space has n^2-n neighbours, where n is the number of nodes in the input graph. 2-opt does not partition the search space, meaning it is always possible for one to get from any state to any other state with a succession of 2-opt operations. Such a property is very desirable because it makes always possible to get to the global minimum regardless of our starting position. We adopt first improvement because, as explained in Section 1.2, we need the indeterminism that first improvement provides for both GWTW and SR.

All the algorithms introduced so far require the construction of root nodes. The GRASP meta-heuristic specifically uses a greedy-randomized heuristic, the GWTW and SR algorithms state nothing about the root node, so the conclusions in Aldous and Vazirani (1994) should hold regardless of generating scheme we implement. The root node, however, defines the topology of the tree we will search in GWTW and SR, so is an important choice. To keep this work consistent with our motivating paper we will use the same root for both.

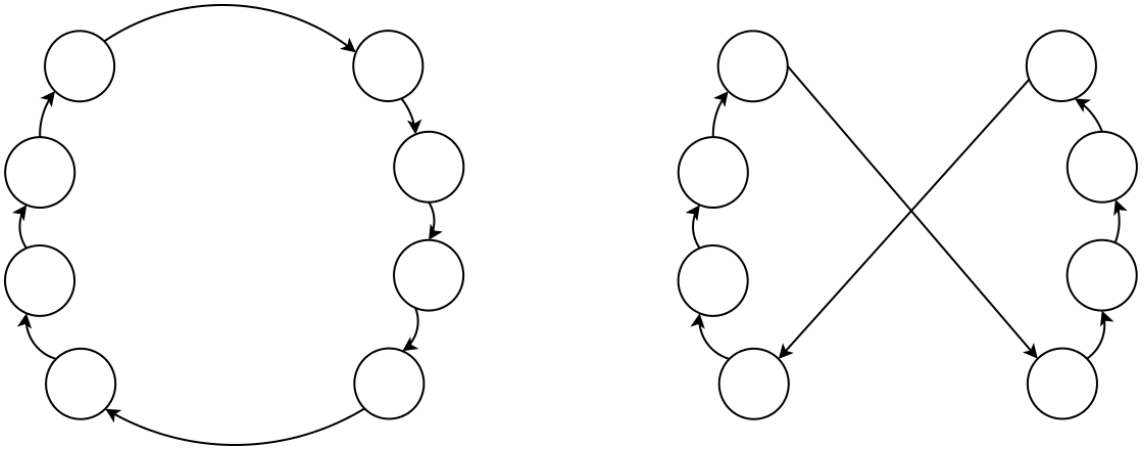


Figure 2.1: An example of a 2-opt operation.

2.4 Results

2.4.1 Experimental Setup

In order to evaluate our algorithms we used the set of instances in *TSPLIB* (2013). It is a commonly used source to benchmark the efficiency of algorithms for the TSP. In the set we have instances with a wide range of sizes, from 14 to 85900 nodes. In the same source we have also the best known cycle for each of the instances, with its associated cost. The solution for some of the biggest instances are heuristic, but most of them are guaranteed global minima.

In the following sections we present several benchmarks, we used a PC with an Intel Core i7 930 processor with 12 GB of main memory for the experiments.

2.4.2 Preliminary Study

The GRASP meta-heuristic depends on the parameter α (not to be confused with α in our $T^{\alpha}_{m,n}$ family), which defines how the root solutions will be at each iteration.

The GWTW algorithm depends on the number of particles, called the beam size. More particles will tend to give us better solutions but with an increase in computational cost. The algorithm definition has no requirement on how to construct the root node, so we will build the root solutions exactly like in grasp making GWTW dependent on the parameter α as well.

The SR serves as our baseline for comparison. In order to keep our work faithful to the theoretical predictions in the motivating paper we will build the root of simple restart like GWTW, including the value for α.

We performed a preliminary set of tests with a small subset of the maps available, for each of the maps we ran each algorithm configuration four times with different random seeds. Each measurement in the following is the mean value for all runs. We stopped all algorithms in thirty minutes and while GWTW could stop earlier, in the case of the preliminary test it was always before the time ran out.

Tables 2.1 and 2.2 contain the results of the preliminary tests to adjust the parameters. Best cost is the best value known, as found in *TSPLIB* (2013). The column "Cost" is the value found in the particular run. "T_{best}" and "T_{finish}" are the time in seconds the algorithm took to find the best solution and to finish respectively.

Table 2.1: Preliminary testing for Go With The Winners

Name	Cities	Best Cost	Cost	T_{best}	T_{finish}
$beam = 16$, greedy solution					
a280	280	2579	2727	0	0
d493	493	35002	36742	1	2
dsj1000	1000	18659688	20147754	16	22
fl3795	3795	28772	30748	166	222
$beam = 16$, random solution					
a280	280	2579	2847	1	1
d493.tsp	493	35002	37847	2	5
dsj1000	1000	18659688	20548185	11	26
fl3795	3795	28772	32126	196	267
$beam = 16$, $\alpha = 10$					
a280	280	2579	2824	1	1
d493	493	35002	37744	1	3
dsj1000	1000	18659688	20582448	25	29
fl3795	3795	28772	32170	198	310
$beam = 4$, greedy solution					
a280	280	2579	2765	0	0
d493	493	35002	37215	0	1
dsj1000	1000	18659688	20357360	2	3
fl3795	3795	28772	31255	21	26
$beam = 4$, random solution					
a280	280	2579	2902	0	0
d493	493	35002	38238	0	1
dsj1000	1000	18659688	20655868	6	10
fl3795	3795	28772	32075	87	98
$beam = 4$, $\alpha = 10$					
a280	280	2579	2895	0	0
d493	493	35002	38340	0	1
dsj1000	1000	18659688	20961455	2	5
fl3795	3795	28772	32957	47	61

Table 2.2: Preliminary testing for GRASP

Name	Cities	Best Cost	Cost	T_{best}
		$\alpha = 2$		
a280	280	2579	2734	942
d493	493	35002	37167	806
dsj1000	1000	18659688	20382143	1066
fl3795	3795	28772	32197	877
		$\alpha = 3$		
a280	280	2579	2720	815
d493	493	35002	37102	1195
dsj1000	1000	18659688	20366442	1120
fl3795	3795	28772	31928	744
		$\alpha = 5$		
a280	280	2579	2711	420
d493	493	35002	37202	1068
dsj1000	1000	18659688	20414262	1260
fl3795	3795	28772	32014	850
		$\alpha = 10$		
a280	280	2579	2695	1266
d493	493	35002	37013	1345
dsj1000	1000	18659688	20264846	883
fl3795	3795	28772	31679	516
		$\alpha = 15$		
a280	280	2579	2740	1245
d493	493	35002	37175	802
dsj1000	1000	18659688	20361730	886
fl3795	3795	28772	31938	929
		$\alpha = 20$		
a280	280	2579	2735	539
d493	493	35002	37148	669
dsj1000	1000	18659688	20409885	1231
fl3795	3795	28772	32122	923

2.4.3 Comprehensive Study

Using the preliminary experimentation as a guideline we performed a comprehensive experiment on all maps available in our benchmark set. We used the parameter configuration that yielded the best results in the preliminary experiments: GRASP with $\alpha = 10$, GWTW with completely greedy start and beam size 16. As explained we decided to adopt the same α for the simple restart as in the GWTW algorithm.

It is not straightforward to compare best values found for the algorithms in a given time. The GWTW has a natural stopping point while the other two can go on forever. We can see in the preliminary tests that the GWTW tends to find the best solutions among the algorithms in consideration but a difference in cost found does not translate directly on how much "effort" the other algorithms would need in order to get to that better value. If we are very near an optimal solution it may be very hard to improve, while if we have a bad solution it will probably be easier to improve by a large margin.

To get a good indication if there are situations where the GWTW clearly outperforms the other algorithms, we run the GWTW get the best result it finds and run the other algorithms until they found an answer that is as good or better. Since there are some big instances in the set we had to give a maximum of twenty minutes for the GWTW and one hour to the other ones.

We ran each algorithm four times for each map and extracted the mean for each measurement. The Tables with the results are in appendix A. The semantics of all columns are the same, we introduce the measurement "%" which is $100 \times \frac{Cost - Best\ Cost}{Best\ Cost}$ and the columns "Transpos" and "Searches". "Trasnpos" appears in tables related to GWTW and count the number of times a particle got into a local minimum and was put in the state of another one. "Searches" appears in tables related both to SR and GRASP and count the number of local searches performed.

2.4.4 Conclusion

In Figure 2.2 we plot the time both GRASP and SR took to find a solution as good as GWTW. We use the time GWTW took to finish as the unit of time, so a normalized time of x for a given algorithm means it is x times slower than GWTW (faster if $x < 1$). Only maps were the GWTW finished are plotted. In most of the maps with more than 1000 cities both GRASP and the simple restart weren't able find a solution as good as the GWTW, we still plotted their time to finish, marked as "unfinished" in the figures.

We can see in the figures that the smallest instances do not indicate much distinction between techniques, that happens because they have such a small search space that a few local searches already find optimal solutions. The maps with more than fifty nodes show a considerable advantage for the GWTW, consistently finishing fifty times faster or better than the other algorithms. It is important to point out that in the bigger instances, where we have the "unfinished" points there is the effect of the limited time, they never got to a solutions as good as GWTW even with much time available.

Time is not completely reliable as a measurement because it is dependent on the particular implementation, an alternative to measure time is to define a basic operation and compare how many each algorithm executed, this measurement makes the evaluation independent on the details of implementation.

The number of local searches is a natural candidate for basic operation for both GRASP and SR, yet the GWTW does not perform isolated local searches. We can, however, count the times a particle reaches a local minimum and is put into the state of a still

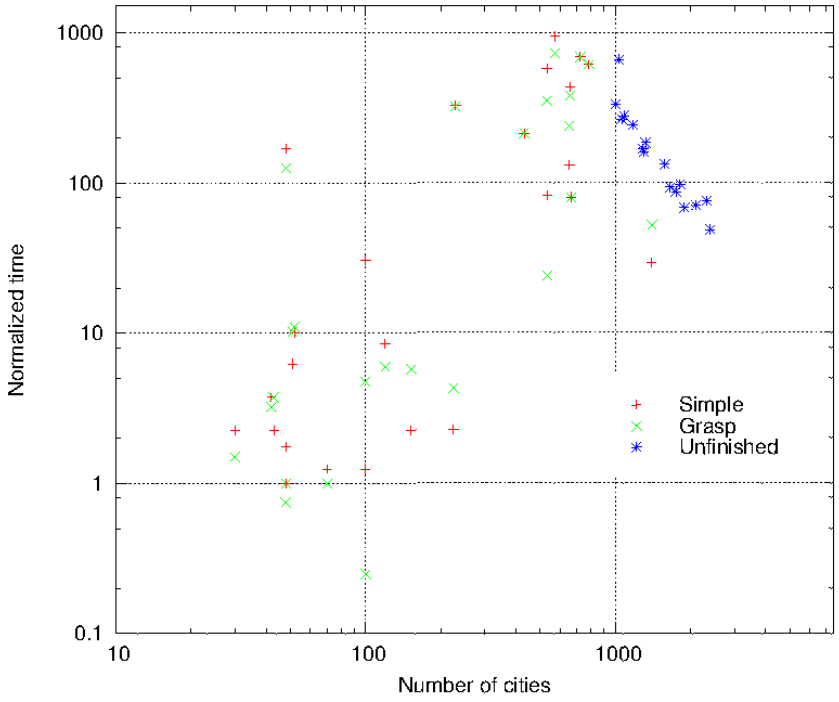


Figure 2.2: Time each algorithm took to finish in relation to GWTW's time by the number of cities.

live particle, the column "Trasnpos" in our tables. A particle life cycle is equivalent to the local searches in the other two algorithms, they won't be exactly equally computing intensive since the particles start in very specific positions, so while it this metric is not a perfect unit of measure, it carries some significance.

In Figure 2.3 we use our metric, counting local searches and particle life cycles to evaluate the performance of the algorithms. In the same fashion of the Graphic 2.2 we use the number of steps GWTW took as unit of measure, the vertical axis is the number of steps took by the algorithm in question divided by the number of steps took by GWTW. One can see the same rough shape in this graph reinforcing our previous analysis.We have a confirmation of the theoretical predictions, but also the comparison with GRASP is also favorable which was not strictly covered by the theoretical framework.

We cannot say we have a strictly better technique than GRASP, SR and even a simple local search as well. We worked in the time frame of the GWTW, if we need a quick answer a local search will be better, if we do not reach a satisfiable answer during the execution of the GWTW we have no option to continue improving while GRASP can continue to run, although with diminishing returns, until you have a good enough solution.

For the travelling sales person problems our test indicate that the GWTW can considerably outperform the GRASP meta-heuristic in the time scale it works best. We can assert that GWTW should be considered as as a viable self contained algorithm. We can also conjecture that it might be viable as a substitute of the local search of a meta-heuristic in similar problems depending on the time frame we can expect the algorithm to run.

It is important to highlight we drew all these conclusions based on the TSP with the 2-opt neighbourhood. We do not know if the results hold for different search spaces topologies. We perform further experiments to partially answer this open question.

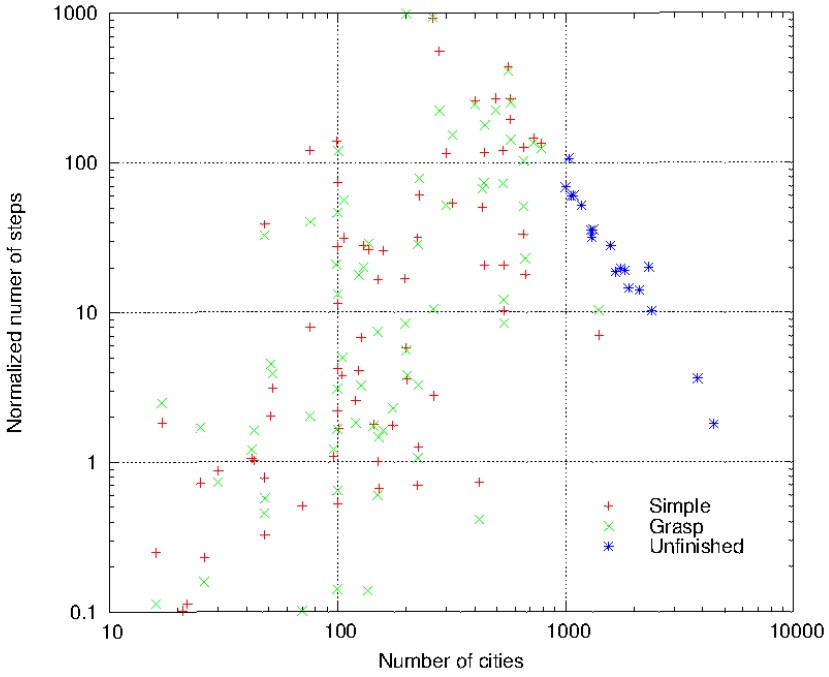


Figure 2.3: Proportion of local searches performed by each algorithm by the number of cities.

3 MACHINE REASSIGNMENT

3.1 Introduction

The second application of GWTW we will investigate is the machine reassignment (MR) problem. Unlike TSP, it is a modern problem, and not as well studied and understood. MR can be an interesting complement to our first study, it is an NP-Hard problem (Gabriel Marques Portal 2012) and the GWTW is in principle a suitable strategy to tackle it.

The main elements are the machines, processes and resources. A valid solution is an assignment of processes to machines that has to satisfy a set of hard constraints. The objective is to improve the usage of the machine resources the best way possible. Initially we are given a valid assignment and new assignment costs depend on this initial state, because moving the processes has an associated cost.

Each instance defines services which represent an actual applications, each service is the set of the processes that constitute the application. Locations are sets of machines, machines in the same location are geographically close. Neighbourhoods are also sets of machines, machines that share a neighbourhood have efficient communication with each other.

The problem was proposed in *ROADEF/EURO CHALLENGE* (2012), it models a practical problem that arises at Google. To gauge GWTW's performance we will compare it with our baseline algorithm SR and the well known simulated annealing (SA) metaheuristic.

3.2 Problem Definition

Let M be the set of machines and P the set of precesses, a valid solution is a function : $P \mapsto M$ which obeys a set hard constraints. We are given an initial solution S_0, which is always valid, and it is important because we have costs associated with the reassignment of processes to different machines.

3.2.1 Hard Constraints

Capacity constraints: Capacity constraints guarantee that the usage of the resources does not exceed the capacity of the machines to provide it. Let R be the set of resources, $C(m, r)$ the capacity of resource $r \in R$ for machine $m \in M$ and $R(p, r)$ the requirement of resource $r \in R$ in process $p \in P$.

The usage of a resource $r \in R$ in a machine $m \in M$. The usage $U(m, r)$ for a given mapping is then $U(m, r) = \sum_{p \in P \wedge M(p) = m} R(p, r)$. The capacity constraint is the require-

ment that each machine must have all the resources available to execute all processes we assign to it, as in equation 3.1.

$$\forall m \in M, r \in R \left(C(m,r) \geqslant R(p,r)\right) \tag{3.1}$$

Conflict constraints: Conflict constraints require that two processes from the same services run in different machines. We want to avoid that services loose more than one process if a machine fails. Let S be the set of services, each service $s \in S$ is a set of processes that must run on distinct machines. All services are disjoint, meaning a process will not be part of two distinct services. We see the formal definition of the conflict constraint in equation 3.2.

$$\forall s \in S, (p_i, p_j) \in s^2, p_i \neq p_j \Rightarrow M(p_i) \neq M(p_j) \tag{3.2}$$

Spread constraints: Let L be the set of locations. Each location is a set of machines. All locations are disjoint, so machines can not be in two distinct locations. For each $s \in S$ we have $spreadMin(s) \in \mathbb{N}$ which gives us the minimum number of distinct locations where at least one process of service s should run, as shown in equation 3.3.

$$\forall s \in S, \sum_{l \in L} min\left(1, \left|\{p \in s | M(p) \in L\}\right|\right) \tag{3.3}$$

Dependency constraints: Let N be the set of neighbourhoods where a neighbourhood is a set of machines, which are disjoint. If a service s^a depends on another service s^b then a process of s^b must run on the neighbourhood of s^a. The formal definition of this constraint is in equation 3.4.

$$\begin{aligned} \left(s^a \in S \text{ depends on } s^b \in S\right) \rightarrow \\ \left(\forall p^a \in s^a, \exists p^b \in s^b, \exists n \in N, (M(p^a) \in n) \wedge (M(p^b) \in n)\right) \end{aligned} \tag{3.4}$$

Transient usage constraints: If we move a process from one machine to another we might need to allocate its resources twice. An example is memory, we may need to reserve the memory to keep the information about the process in both machines until the transfer is completed. Let $TR \subseteq R$ be the subset of resources which need transient usage then equation 3.5 defines the transient usage.

$$\forall m \in M, r \in TR, \sum_{\substack{p \in P \text{ such that} \\ M(p)=m \vee S_0(p)=m}} R(p,r) \leqslant C(m,r) \tag{3.5}$$

3.2.2 Costs

We next define the objective function, we will first present each different cost alone and then access the overall objective function.

Load cost: Let $SC(m,r)$ be the safety capacity of a resource $r \in R$ on a machine $m \in M$. We prefer solutions that processes use the least possible resources above the safety capacity, as in Equation 3.6

$$loadCost(r) = \sum_{m \in M} \max(0, U(m,r) - SC(m,r)) \tag{3.6}$$

Balance cost: It is usual to use resources in conjunction, for instance memory and CPU, therefore we prefer that the resource usage is balanced so future assignments are easier to allocate. Let B be a set of triples defined in $\mathbb{N} \times R^2$, for a triple $b \leqslant r_1, r_2, target >\in B$, the balance cost is than given by Equation 3.7b.

$$A(m, r) = C(m, r) - U(m, r) \tag{3.7a}$$

$$balanceCost(r) = \sum_{m \in M} \max(0, target.A(m, r_1) - A(m, r_2)) \tag{3.7b}$$

Process move cost: Not all processes are equally easy to move, some might be very easy while others very hard. Let $PMC(p)$ be to cost to move process p from its original machine $M_0(p)$. The process move is then given by equation 3.8.

$$processMoveCost(r) = \sum_{\substack{p \in P \text{ such that} \\ M(p) \neq M_0(p)}} PMC(p) \tag{3.8}$$

Service move cost: Service move cost is the maximum number of moved processes over services, defined in equation 3.9.

$$serviceMoveCost(r) = \max\left(\left|\{p \in s | M(p) \neq S_0(p)\}\right|\right) \tag{3.9}$$

Machine move cost: Let $MMC(m_{source}, m_{destination})$ with $m_{source}, m_{destination} \in M$ be the cost of moving any process between the two machines, the machine move cost is given then by equation 3.10

$$machineMoveCost = \sum_{p \in P} MMC(M0(p), M(p)) \tag{3.10}$$

Total objective function: The total objective function is the weighted sum of all its components, as shown in equation 3.11.

$$\begin{aligned} totalCost = & \sum_{r \in R} weightloadCost(r) \cdot loadCost(r) \\ & + \sum_{b \in B} weightbalanceCost(b) \cdot balanceCost(b) \\ & + weightprocessMoveCost \cdot processMoveCost \\ & + weightserviceMoveCost \cdot serviceMoveCost \\ & + weightmachineMoveCost \cdot machineMoveCost \end{aligned} \tag{3.11}$$

3.3 Machine Reassignment Example

To illustrate the definition above we will present a minimalistic example of an instance of the problem.

3.3.1 Instance definition

In our example instance we have the following elements:

- Machines: $\{m_1, m_2, m_3\}$

- Resources: $\{r_1\}$, where r_1 has no transient usage. Every machine has the capacity of 10 units and safe capacity of 5 units of r_1.
- Precesses: $\{p_1, p_2, p_3, p_4\}$, where p_1, p_2, p_3 and p_4 require 3, 4, 5 and 6 units of the resource respectively.
- Services: $\{s_1, s_2\}$, where $s_1 = \{p_1, p_2\}$ and $s_2 = \{p_3, p_4\}$ and s_2 depends on s_1.
- Locations: $\{l_1, l_2, l_3\}$, where $l_1 = \{m_1\}$, $l_2 = \{m_2\}$ and $l_3 = \{m_3\}$.
- Neighbourhoods: $\{n_1\}$, where n_1 trivially contains all machines.

The minimum spread is defined as 2. The balance cost is trivially zero since we have only one resource. The move costs is 0.5 for all machines, 0.5 for all processes and the service move cost is always 0.

The weights in Equation 3.11 are all 1.

The initial assignment is p_1 and p_3 to m_1, p_2 to m_2 and p_4 to m_3, which is a valid assignment since it obeys al hard constraints:

- Capacity constraint: m_1 uses 8, m_2 uses 4 and $m3$ uses 6 units of r_1, which is beneath their maximum load.
- Conflict constraint: Only p_1 and p_3 are in the same machine and they are from different services so there is no conflict.
- Spread constraint: Each service is running in the two locations so we have the minimum required spread of 2.
- Dependency and transient usage: Are satisfied since we have only one neighbourhood and no transient usage.

The cost of this assignment is:

- Load cost is $\max\big(0, (8-5)\big) + \max\big(0, 4-5)\big) + \max\big(0, (6-5)\big) = 4$
- Balance cost is 0 since we have only one resource.
- All move costs are 0 since it is the initial assignment.

The total cost of the initial assignment is then 4.

3.3.2 New Assignment

We can generate a new valid assignment by moving p_3 from m_1 to m_2, so the new assignment is p_1 to m_1, p_2 and p_3 to m_2 and p_4 to m_3. We can see that this movement produces a state that still obeys the hard constraints:

- Capacity constraint: m_1 uses 3, m_2 uses 9 and m_3 uses 6 units of r_1.
- Conflict constraint: Only p_2 and p_3 are in the same machine and they are from different services so there is no conflict.
- Spread constraint: Each service is running in the two locations so we have the minimum required spread of 2.

- Dependency and transient usage: Are satisfied since we have only one neighbourhood and no transient usage.

The cost of this assignment is:

- Load cost is $\max\big(0,(3-5)\big)+\max\big(0,9-5)\big)+\max\big(0,(6-5)\big)=5$
- Balance cost is 0 since we have only one resource.
- We have the move costs of transferring p_3 from m_1 to m_3 which is $0.5+0.5+0=1$.

The total cost of this new assignment is then 6.

3.4 Implementation

To prepare our implementation to test GWTW we adapted the simulated annealing solver code of the machine reassignment problem. The precise description of the algorithm can be found in Gabriel Marques Portal (2012). It was originally submitted in ROADEF/EURO challenge 2011-2012 and achieved forth position.

The GWTW requires that we keep the information about more than one state in the search space at the same time, and the ability to transform one state into another. We adapted the data structure to perform the transformation by overwriting the state information of one state by another. In this implementation of the machine reassignment problem this operation is computationally demanding since the state information and meta-data can be large. The meta-data is important because it improves the speed of the transition to neighbours that is the most common operation.

3.4.1 Neighbourhood

We adapted the original implementation to compute the state transitions, but we kept the same definition of neighbourhood. We compute a neighbour by performing either a *swap* or a *shift*. A shift moves a single process from one machine to another and a swap exchanges two processes in different machines.

Let m be the number of machines and p be the number of processes of a particular instance. The number of potential neighbours of a particular state is $p\times(m-1)$ via shift move and $p\times(p-1)$ via swap move. In the harder instances of our test set this number is big enough that it is impractical to traverse all the potential neighbourhood. We avoid searching all the neighbourhood and search a manageable subset of the possible neighbours.

It is important to point out that neither all shifts nor all swaps guarantee to generate a valid state, changing the allocation of processes we may break a hard constraint generating an invalid state. When we need to compute a neighbour of a state we need to validate the new state, a computationally expensive process.

Computing the cost of a given state from scratch is expensive, so we maintain the original SA implementation and compute it dynamically, meaning we compute the difference between the current state cost and the new state cost.

In Algorithm 6 we see the pseudo-code of the neighbourhood search. We begin choosing randomly, with probability α, if we will attempt to shift or swap. If the move fails we try it up to τ more times, if we find no new valid states that improve our current solution we assume we are in a local minimum.

To perform a shift, whose pseudo-code is in Algorithm 7, we randomly select a process and a machine. We verify if the new state is valid and if the cost is better than the current one. If it is, than we return the newly generated state. If it is not, we try again, up to n times, with the next machine. If none of the attempts generate a suitable new state, we declare that we failed to perform the movement. Notice that if Algorithm 6 attempts to move again it may work because we will start, with high probability, with two different random starting process/machine pairs. A swap is similar to a shift but the second element of the pair is another process, with which we hope to swap, instead of a machine.

Algorithm 6 Neighbourhood

Parameter α : Chance to shift instead of swap.
Parameter τ : Maximum new random candidate states.
Parameter n : Maximum states we search given a starting candidate.
for τ times **do**
  $r = random\ number \in [0, 1]$
  if $\alpha > r$ **then**
    $shift(n)$
  else
    $swap(n)$
  end if
end for

Algorithm 7 Shift

Parameter n : Maximum states we search given a starting candidate.
Randomly select a process p_i and a machine m_j.
for offset $= 0 \ldots n$ **do**
  if Shift of p_i to $m_{j+\text{offset}}$ generates a valid state. **then**
    Compute cost of state with p_i mapped to $m_{j+\text{offset}}$
    if New cost is better than current one. **then**
      Execute shift of p_i to $m_{j+\text{offset}}$
    end if
  end if
end for

Algorithm 8 Swap

Parameter n : Maximum states we search given a starting candidate.
Randomly select two processes p_i and p_j.
for offset $= 0 \ldots n$ **do**
  if Swap of p_i and $p_{j+\text{offset}}$ generates a valid state. **then**
    Compute cost of state with p_i swapped with $p_{j+offset}$
    if New cost is better than current one. **then**
      Execute swap of p_i to $p_{j+\text{offset}}$
    end if
  end if
end for

Our method for generating neighbours has the particularity that it may fail to generate a new state even if we are not in a local minimum. Since we randomly choose a subset of the possible neighbours to investigate we may miss a possibility to improve the current objective function value. Both GWTW and SR depend on the identification of local minima. The parameters τ and n define how many attempts, $\tau \times n$, to find a new valid state that improves the current one we perform before assuming we are in a local minimum.

3.4.2 Simulated Annealing Implementation

In Algorithm 9 we have the the modified simulated annealing used in the competition and we use to compare the quality of the solutions GWTW finds. We start with a given temperature and try to perform moves while decreasing the temperature. We count the failures and successes, resetting the temperature if it fails too often. The movement is described in Algorithm 10. It is similar to the one described in our neighbourhood but it may allow a move to a worse state than the current one, depending on the temperature and decrease of the quality of the solution.

Algorithm 9 Simulated Annealing

Parameter T_0 : Initial temperature.
Parameter α : Chance to shift instead of swap.
Parameter τ : Maximum new random candidate states.
Parameter n : Maximum states we search given a starting candidate.
Parameter η : minimal percentage to not increase $freezeCount$
Parameter ω : $freezeCount$ threshold to activate re-heating
Parameter r : Rate of cooling.
Current temperature $T_{curr} = T_0$
while Stopping criterion not satisfied **do**
  for n times **do**
    Attempt $SAmove$
    if New best solution found **then**
      $freezeCount = 0$
    end if
  end for
  if Percentage of $SAmove$ success is below η **then**
    Add 1 to $freezeCount$.
    if $freezeCount$ over threshold of ω **then**
      re-heat: $T_{curr} = T_0/100$
    end if
  end if
  $T = r \times T$
end while

Algorithm 10 SAMove

Parameter α : Chance to shift instead of swap.
Parameter τ : Maximum new random candidate states.
Parameter n : Maximum states we search given a starting candidate.
Parameter T : Current temperature.
for τ times **do**
  $r = random\ number \in [0, 1]$
  if $\alpha > r$ **then**
    $SAshift(n)$
  else
    $SAswap(n)$
  end if
end for

Algorithm 11 SAshift

Parameter n : Maximum states we search given a starting candidate.
Parameter T : Current temperature.
Randomly select a process p_i and a machine m_j.
for offset $= 0 \ldots n$ **do**
  if Shift of p_i to $m_{j+\text{offset}}$ generates a valid state. **then**
    Compute cost of state with p_i mapped to $m_{j+\text{offset}}$.
    $\delta =$ new cost $-$ current cost.
    if New cost is better than current one, or with chance $e^{\delta/T}$ **then**
      Execute shift of p_i to $m_{j+\text{offset}}$.
    end if
  end if
end for

Algorithm 12 SAswap

Parameter n : Maximum states we search given a starting candidate.
Parameter T : Current temperature.
Randomly select two processes p_i and p_j.
for offset $= 0 \ldots n$ **do**
  if Swap of p_i and $p_{j+\text{offset}}$ generates a valid state. **then**
    Compute cost of state with p_i mapped to $m_{j+\text{offset}}$.
    $\delta =$ new cost $-$ current cost.
    if New cost is better than current one, or with chance $e^{\delta/T}$ **then**
      Execute swap of p_i to $m_{j+\text{offset}}$.
    end if
  end if
end for

Table 3.1: Information on the test set.

Name	Initial Solution	Processes	Machines
A1 1	49528750	100	4
A1 2	1061649570	1000	100
A1 3	583662270	1000	100
A1 4	632499600	1000	50
A1 5	782189690	1000	12
A2 1	391189190	1000	100
A2 2	1876768120	1000	100
A2 3	2272487840	1000	100
A2 4	3223516130	1000	50
A2 5	787355300	1000	50
B1	7644173180	5000	100
B2	5181493830	5000	100
B3	6336834660	20000	100
B4	9209576380	20000	500
B5	12426813010	40000	100
B6	12749861240	40000	200
B7	37946901700	40000	4000
B8	14068207250	50000	100
B9	23234641520	50000	1000
B10	42220868760	50000	5000

3.5 Results

3.5.1 Experimental Setup

We will compare GWTW, simple restart and simulated annealing. For the simulated annealing we used the values $\alpha = 0.3$, $n = 10^3$ and $I = 10^3$ used in the original implementation since they were adjusted to compete in the ROADEF Challenge 2011/2012.

For the SR and GWTW, we have the parameters α and n which are directly analogous to parameters in Simulated Annealing so we will use the same values. τ has very similar semantics in GWTW, SR and SA. In both GWTW and SR it is related to how long we persist before assuming we are in a local minimum. In SA it is related to how long we take to cool down the current temperature, we will use also the same value e SA. The only parameter left to adjust is the beam size.

3.5.2 Instances

The ROADEF Challenge 2011/2012 released three sets of benchmark instances. All sets together contain twenty instances, which vary in size and difficulty, details on the instances can be seen in Table 3.1.

In the original definition of the problem we have the specification that the candidates would run five minutes. We decided to use this time in all our benchmarks, since it was part of the specification of the problem.

3.5.3 Parameter Adjusting

We first use a subset of the problems to calibrate the beam size of GWTW. We chose one instance from each set. The results are in Tables 3.2, 3.3 and 3.4.

In the column “Algorithm” we have the type of algorithm, each algorithm has being

Table 3.2: Preliminary testing of Go With The Winners. Instance A1 3

Algorithm	Beam Size	Cost	%	T_{finish}
SA		583008879	0.0	300
SR		583006164	0.0	300
GWTW	2	583128679	0.2	0.0
GWTW	4	583020384	0.0	0.1
GWTW	6	583008638	0.0	0.1
GWTW	8	583007534	0.0	0.1

Best cost found: 583006114.

Table 3.3: Preliminary testing of Go With The Winners. Instance A2 3

Algorithm	Beam Size	Cost	%	T_{finish}
SA		1308023378	2	300
SR		1585895346	29	300
GWTW	2	1663114155	25	1
GWTW	4	1644443537	24	2
GWTW	6	1629370678	22	3
GWTW	8	1623150705	22	5

Best cost found: 1285662633.

Table 3.4: Preliminary testing of Go With The Winners. Instance B 6

Algorithm	Beam Size	Cost	%	T_{finish}
SA		9525865267	0	300
SR		10878103518	14	300
GWTW	2	9776522233	4	3
GWTW	4	9526345346	0	10
GWTW	6	9526278046	0	20
GWTW	8	9526228676	0	47

Best cost found: 9525859126.

executed four times in each instance and we extracted the mean value for each measurement. The column "%" indicates the percentage above the best cost found for the particular instance. "T_{best}" and "T_{finish}" indicate the time in seconds the algorithm took to find the best solution and to finish respectively. Notice that since neither SA nor SR have a stopping point besides the time so "T_{finish}" is always 300.

As expected we have an increase in the quality, with diminishing returns, with an increase in execution time as we increase beam size. We choose to investigate the beam with four because it has a big advantage over a smaller beam size of two: From beam with size two to four we have a mean improve of 1% on the solution while the other improves from the increase in beam are much smaller. We also test a beam size of eight because it is the number that generated the best solutions, and while it has higher processing time it is well under our limit of five minutes.

3.5.4 Go With The Winners Test

We performed a comprehensive set of tests of GWTW with beam size four and eigh, and also SA and SR. In Tables 3.5 and 3.6 we see the results of our tests for beam size four and eight in each table respectively. We performed four executions with different seeds each had a maximum of five minutes. The tests were performed in an Intel Core i7 3.40 GHz. The column "Best" is the best value we found for that instance. "GWTW", "SA" and "SR" represent the mean value that particular algorithm found. The columns "%", "SA%" and "SR%" represent the percent difference between the cost found by GWTW and the best cost. We have the time, in seconds, GWTW took to finish in the column "t", we have not included SA's and SR's time to finish because it is trivially five minutes.

The first noticeable things in the both tables are the values for instance A2 1. SA achieves a solution several orders of magnitude better than SR and GWTW. We think it is because GWTW does not allow us to worsen the current solution and scape local minima, we have a limited part of the search space to look for a good solution. It is likely, that a larger beam will not help achieve a comparable value to SA. We do not take this outlier into account in our averages so it does not distort the other results.

From both Tables 3.5 and 3.6 we compute the average for all maps of the % column for each algorithm: $\overline{SA} = 0.8\%$, $\overline{GWTW8} = 12\%$, $\overline{GWTW4} = 14\%$ and $\overline{SR} = 31.9\%$.

GWTW systematically achieved better results than SR, regardless of beam size. The averages computed above are smaller for GWTW than for SR which means it achieves closer results to the best one found. GWTW found better solution than SR 13 times of the possible 20, with a beam size of four and eight. The times these algorithms ran is not the same, GWTW used in average 11.5 seconds while GWTW 8 51.7 seconds, so this comparison is not completely fair for GWTW. With this difference in the quality of the solutions and running time we have another indication that the advantages in the GWTW in relation to SR can be translated to actual optimization problems.

The algorithm with beam size eight achieves better solutions in average, which can be quantified by $\overline{GWTW4} - \overline{GWTW8} = 2\%$, but ran in average 4.5 times longer. GWTW with beam size eight found a better solution than with beam size four in every instance. Gwtw with beam size four gets solution close to the one with beam size eight faster but they are always worse solutions, we cannot conclude that one is strictly better than the other.

The SA achieved the best results among the algorithms we tested, $\overline{SA}$ is the lowest, when compared to $\overline{GWTW8}$, $\overline{GWTW4}$ and $\overline{SR}$, it is 11.2% better than the second best $\overline{GWTW8}$. SA also got the best value in 18 of the 20 instances, while GWTW with beam

Table 3.5: Test of Go With The Winners with beam size four.

Name	Best	GWTW	%	t	SA	SA%	SR	SR%
A1 1	44306501	44306677	0	0	44306985	0	44306501	0
A1 2	777533098	800658245	2.9	0	777533374	0	785363472	1
A1 3	583006114	583020384	0	0	583008879	0	583006164	0
A1 4	261584325	286171725	9.3	1	264797606	1.2	270470517	3.3
A1 5	727578310	727580613	0	0	727578310	0	727579131	0
A2 1	221	5538055	$\sim 10^6$	0	223	1	1957476	$\sim 10^6$
A2 2	838768542	1252701865	49.3	1	876388209	4.4	1290690624	53.8
A2 3	1285662633	1644443537	27.9	2	1308023378	1.7	1585895346	23.3
A2 4	1680514683	1856807482	10.4	1	1680565446	0	1787005740	6.3
A2 5	310793983	501354708	61.3	1	314620481	1.2	485586476	56.2
B 1	3433161423	4192297903	22.1	9	3449932625	0.4	4185033089	21.9
B 2	1015703411	1124838504	10.7	4	1015766566	0	1182007836	16.3
B 3	156829032	244878190	56.1	25	156850597	0	298262906	90.1
B 4	4677888207	4678753998	0	7	4677892969	0	4679056336	0
B 5	923045507	1057497984	14.5	28	923059780	0	1113063952	20.5
B 6	9525859126	9526345346	0	10	9525861550	0	9526621214	0
B 7	14835071345	14925620895	1	18	14835556720	0	37389950536	152
B 8	1214510093	1216720403	0.1	42	1214544002	0	1240967902	2.1
B 9	15885762453	15887127138	0	15	15885766417	0	15887351324	0
B 10	18048514734	18098832656	0.3	56	18048530008	0	41598085459	130.4

size 8 got the best value twice. SA achieved better results overall than GWTW but it is important to remember the time difference, and that this is a first approach to implement GWTW. Considering that GWTW finished much faster than SA and that the results it produced were not far from SA we think it have another indication it is worth to continue to investigate the GWTW approach.

3.5.5 Iterative Go With The Winners Test

We will also investigate the algorithm we call *Iterative Go With The Winners* (IGWTW). It is an attempt to develop a simple extension to GWTW that runs an arbitrary amount of time. IGWTW repeatedly executes a simple GWTW runs, for increasing beam sizes. It is specified by an initial beam size, and an increment. IGWTW n m means that the initial beam size is n, and the beam size is increased by m. The procedure is defined in Algorithm 13. It consists on performing repeatedly the GWTW algorithm. The first time with the starting beam size, and increase it each run. It is not a fully developed meta-heuristic just a way to adapt the algorithm to run all five minutes regardless of the instance.

Algorithm 13 Iterative Go With The Winners

Parameter β : Basic beam size.
Parameter ι : Beam size increase.
$b = \beta$
while Still have time **do**
  Run GWTW with beam size b.
  $b = b + \iota$
end while

We used the base of our IGWTW as four because it had good performance in our

Table 3.6: Test of Go With The Winners with beam size eight.

Name	Best	GWTW	%	t	SA	SA%	SR	SR%
A1 1	44306501	44306602	0	0	44306985	0	44306501	0
A1 2	777533098	796234147	2.4	2	777533374	0	785363472	1
A1 3	583006114	583007533	0	0	583008879	0	583006164	0
A1 4	261584325	280166030	7.1	3	264797606	1.2	270470517	3.3
A1 5	727578310	727580460	0	0	727578310	0	727579131	0
A2 1	221	5161928	$\sim 10^6$	0	223	1	1957476	$\sim 10^6$
A2 2	838768542	1237851860	47.5	4	876388209	4.4	1290690624	53.8
A2 3	1285662633	1623150705	26.2	5	1308023378	1.7	1585895346	23.3
A2 4	1680514683	1812271848	7.8	4	1680565446	0	1787005740	6.3
A2 5	310793983	482793531	55.3	4	314620481	1.2	485586476	56.2
B 1	3433161423	4086756251	19	33	3449932625	0.4	4185033089	21.9
B 2	1015703411	1081640280	6.4	19	1015766566	0	1182007836	16.3
B 3	156829032	234643298	49.6	89	156850597	0	298262906	90.1
B 4	4677888207	4678453033	0	39	4677892969	0	4679056336	0
B 5	923045507	979618845	6.1	253	923059780	0	1113063952	20.5
B 6	9525859126	9526228676	0	47	9525861550	0	9526621214	0
B 7	14835071345	14836594261	0	95	14835556720	0	37389950536	152
B 8	1214510093	1216433902	0.1	170	1214544002	0	1240967902	2.1
B 9	15885762453	15886801696	0	97	15885766417	0	15887351324	0
B 10	18048514734	18049156383	0	123	18048530008	0	41598085459	130.4

preliminary testing, we increase the beam size by two in each iteration to still maintain the beam size in a manageable size during the run.

We have the results of the experiment with IGWTW in Table3.7. The experimental setting was identical to the previous as is the semantics of all columns. We do not have the run times of any of the algorithms, because the time is always five minutes.

Performing a similar analysis as before we see a small improvement in the quality of solution when comparing IGWTW and GWTW, the mean difference to the best solution found is 8.8% which is an improvement, however SA performed better. Like GWTW with beam size 8 it was able to get better solutions than SA in only two instances. We lost the fast execution time of the pure GWTW. We cannot expect much improvement from the pure GWTW because we still neither allow the search proceed to states that produce worse solutions like SA, nor do we start in different points of the search space like GRASP. We are still confined to search the same states randomized local search can, we just search them more thoroughly.

Table 3.7: Test of Iterative Go With The Winners with base beam size four and increment of two.

Name	Best Found	IGWTW	%	SA	SA%	SR	SR%
A1 1	44306501	44306501	0	44306985	0	44306501	0
A1 2	777533098	786480779	1.1	777533374	0	785363472	1
A1 3	583006114	583006520	0	583008879	0	583006950	0
A1 4	261584325	272158146	4	264797606	1.2	270470517	3.3
A1 5	727578310	727578978	0	727578311	0	727580143	0
A2 1	221	1856462	$\sim 10^5$	223	1	4367281	$\sim 10^5$
A2 2	838768542	1188609001	41.7	876388209	4.4	1290690624	53.8
A2 3	1285662633	1561421257	21.4	1316892088	2.4	1585895346	23.3
A2 4	1680514689	1758106662	4.6	1680565448	0	1787005740	6.3
A2 5	310793983	453543054	45.9	314620481	1.2	485586476	56.2
B 1	3444556602	3989807675	15.8	3452781420	0.2	4185033089	21.9
B 2	1015703411	1056365377	4	1015766566	0	1182007836	16.3
B 3	156829032	198394216	26.5	173481411	10.6	298262906	90.1
B 4	4677889687	4678328685	0	4677894752	0	4679056336	0
B 5	923043348	981693590	6.3	923058208	0	1113063952	20.5
B 6	9525859126	9526080346	0	9525861550	0	9526621214	0
B 7	14835071345	14863397537	0.1	14835556720	0	37389950536	152
B 8	1214510093	1216131047	0.1	1214544002	0	1240967902	2.1
B 9	15885740728	15886737623	0	15885746295	0	15887351324	0
B 10	18048514734	18075113078	0.1	18048530008	0	41598085459	130.4

4 CONCLUSION

We developed a framework to bridge the gap between the theory in Aldous and Vazirani (1994) and actual optimization problems. In it we simulated how both the algorithms SR and GWTW behave, which led us to believe the GWTW could perform well. We, then, implemented the ideas behind GWTW in two problems, the TSP and MR.

In TSP we compared GWTW with both GRASP and SR. SR is our basic algorithm of comparison and GRASP was used to get an effective and commonly used algorithm to compare to. We used the time in which GWTW finishes as our basic time scale and compared to the results from the other algorithms. We had a good performance from GWTW which lead us to believe that in the specific kind of topology and specific time scale it outperforms both algorithms.

We implemented the algorithms GWTW, SR, SA and IGWTW to solve instances of MR. We used the time defined by the problem, five minutes, to perform our tests. GWTW again outperformed SR, reinforcing the idea that the results in the theory can be translated to practical problems. We compared the results of GWTW with SA and SA achieved better results but took more time to do so. We again see the performance of GWTW as a signal it may lead to an effective solver in particular topologies and time scales. The IGWTW was a simple attempt to extend GWTW, but it produced a small improved at cost of execution time. In particular it fails to provide a way to scape local minima which may be the cause that it fails to improve, by a large margin, the pure GWTW.

The results all indicate we have a viable strategy. Since it produced good results in three different topologies we are confident that the approach may work in a good set of different problems. We believe that developing a meta-heuristic based on GWTW or inserting GWTW in meta-heuristic that use local searches in place of them has the potential to bear competitive results depending on the type of problem.

APPENDIX A TSP TABLES

Table A.1: GRASP 1

Name	Cities	Best Cost	Cost	%	T_{best}	T_{finish}	Searches
burma14	14	3323	3323	0	*1	*1	3
ulysses16	16	6859	6859	0	*7	*7	14
gr17	17	2085	2086	0	0	0	14
gr21	21	2707	2707	0	0	0	3
ulysses22	22	7013	7014	0	*114	*114	7
gr24	24	1272	1282	1	*2	*2	10
fri26	26	937	941	0	*2	*2	9
bays29	30	2020	2020	0	*13	*13	44
dantzig42	42	699	708	1	*38	*38	60
swiss42	43	1273	1278	0	*38	*38	57
att48	48	10628	10628	0	*61	*61	39
gr48	48	5046	5088	1	*16	*16	19
hk48	48	11461	11579	1	1	1	1581
eil51	51	426	431	1	*91	*91	83
berlin52	52	7542	7649	1	*227	*227	170
st70	70	675	701	4	*50	*50	21
pr76	76	108159	110080	2	1	1	500
eil76	76	538	551	4	13	13	5951
gr96	96	55209	56547	2	1	1	75
rat99	99	1211	1248	3	23	23	5561
kroE100	100	22068	22664	2	6	6	1240
kroC100	100	20749	21186	2	0	0	130
kroD100	100	21294	22195	4	*196	*196	35
kroB100	100	22141	22440	0	20	20	4276
kroA100	100	21282	21674	1	1	1	300
rd100	100	7910	8181	3	2	2	550
eil101	101	629	655	4	0	0	93
lin105	105	14379	14766	2	1	1	273
pr107	107	44303	44904	1	9	9	1689
gr120	120	6942	7172	3	*950	*950	175
pr124	124	59030	59649	8	1	1	255
bier127	127	118282	121140	2	4	4	573
ch130	130	6110	6338	13	3	13	1627
pr136	136	96772	103503	6	0	0	3
gr137	137	69853	72004	3	129	129	1981
pr144	144	58537	58778	0	0	0	98

* indicates time in milliseconds.

Table A.2: GRASP 2

Name	Cities	Best Cost	Cost	%	T_{best}	T_{finish}	Searches
kroA150	150	26524	27807	4	0	0	58
ch150	150	6528	6665	2	927	927	84787
kroB150	150	26130	26779	2	13	13	1199
pr152	152	73682	75497	2	*611	*611	42
u159	159	42080	43763	3	18	18	1457
si175	175	21407	21509	0	0	0	110
rat195	195	2323	2420	4	1508	2739	155928
d198	198	15780	16110	0	26	26	1284
kroB200	200	29437	31120	5	8	8	425
kroA200	200	29368	30087	2	1717	2856	136824
gr202	202	40160	42105	5	42	42	269
tsp225	225	3916	4133	5	57	57	2364
ts225	225	126643	130235	3	*1056	*1056	34
pr226	226	80369	81684	2	1	1	75
gr229	229	134602	139470	4	429	982	4709
gil262	262	2378	2491	5	411	1850	54007
pr264	264	49135	52362	7	4	4	140
a280	280	2579	2729	6	740	1469	38903
pr299	299	48191	51127	6	457	457	9605
linhp318	318	41345	44236	7	185	185	3335
lin318	318	42029	44236	5	185	185	3335
rd400	400	15281	16199	6	1188	1868	21856
fl417	417	11861	12284	8	5	5	57
gr431	431	171414	180492	5	1449	3600	4426
pr439	439	107217	113545	6	260	260	2366
pcb442	442	50778	54584	7	735	735	7375
d493	493	35002	36912	5	1715	3258	24063
att532	532	27686	28433	3	358	2741	12390
si535	535	48450	48803	1	85	85	731
ali535	535	202339	217069	7	1520	1972	1477
pa561	561	2763	2984	8	2122	3076	29724
u574	574	36905	39797	8	1348	2830	15151
rat575	575	6773	7340	8	1798	3600	20972
p654	654	34643	35508	2	650	650	2574
d657	657	48912	52525	7	1933	2485	9919
gr666	666	294358	318807	8	2045	3600	1742

* indicates time in milliseconds.

Table A.3: GRASP 3

Name	Cities	Best Cost	Cost	%	T_{best}	T_{finish}	Searches
u724	724	41910	45470	8	1872	3600	12209
rat783	783	8806	9637	9	1612	3600	10838
dsj1000	1000	18659688	20316754	9	616	3600	5039
pr1002	1002	259045	282678	9	1235	3600	5931
si1032	1032	92650	93174	1	1325	3600	7144
u1060	1060	224094	244138	9	2592	3600	5144
vm1084	1084	239297	259344	8	1433	3600	4961
pcb1173	1173	56892	62964	10	2218	3600	4380
d1291	1291	50801	57366	12	2679	3600	3351
rl1304	1304	252948	278031	10	1628	3600	3048
rl1323	1323	270199	296951	10	477	3600	3098
fl1400	1400	20127	20954	4	927	927	697
fl1577	1577	22249	24089	8	2200	3600	1953
d1655	1655	62128	69794	12	1057	3600	2101
vm1748	1748	336556	369900	10	2249	3600	1733
u1817	1817	57201	66204	15	1809	3600	1814
rl1889	1889	316536	351430	11	2122	3601	1430
d2103	2103	80450	94176	16	2244	3601	1294
u2319	2319	234256	251490	7	2569	3601	1408
pr2392	2392	378032	426225	12	2163	3600	961
fl3795	3795	28772	31546	10	1551	3605	300
fnl4461	4461	182566	204209	12	1924	3604	277
rl5915	5915	565530	650434	15	1227	3614	128
rl5934	5934	556045	640811	15	2289	3606	126
pla7397	7397	23260728	26194260	13	1360	3616	80
rl11849	11849	923288	1073315	16	2410	3673	28
usa13509	13509	19982859	22486275	13	438	3712	22
brd14051	14051	469385	527857	12	1069	3647	22
d15112	15112	1573084	1767196	12	1841	3724	19
d18512	18512	645238	726239	12	1068	3777	12
pla33810	33810	66048945	76753277	16	3569	4346	2
pla85900	85900	142382641	162389460	14	11054	11054	0

Table A.4: Simple Restart 1

Name	Cities	Best Cost	Cost	%	T_{best}	T_{finish}	Searches
burma14	14	3323	3323	0	*1	*1	2
ulysses16	16	6859	6859	0	*2	*2	6
gr17	17	2085	2085	0	0	0	19
gr21	21	2707	2707	0	0	0	1
ulysses22	22	7013	7014	0	*192	*192	5
gr24	24	1272	1276	0	0	0	2
fri26	26	937	937	0	*1	*1	6
bays29	30	2020	2020	0	*9	*9	37
dantzig42	42	699	706	1	*40	*40	69
swiss42	43	1273	1285	1	*58	*58	91
att48	48	10628	10628	0	*39	*39	28
gr48	48	5046	5099	1	*22	*22	27
hk48	48	11461	11583	1	*1040	*1040	1330
eil51	51	426	432	1	*178	*178	186
berlin52	52	7542	7662	2	*251	*251	211
st70	70	675	699	4	*11	*11	4
pr76	76	108159	109846	2	0	0	127
eil76	76	538	552	3	4	4	1982
gr96	96	55209	56601	3	1	1	84
rat99	99	1211	1242	3	3	3	826
kroE100	100	22068	22798	3	0	0	138
kroC100	100	20749	21309	3	0	0	38
kroD100	100	21294	22125	4	*50	*50	9
kroB100	100	22141	22485	2	12	12	2714
kroA100	100	21282	21676	2	4	4	935
rd100	100	7910	8183	3	*395	*395	80
eil101	101	629	654	4	27	27	6857
lin105	105	14379	14703	2	1	1	361
pr107	107	44303	44780	1	17	17	3066
gr120	120	6942	7190	4	*682	*682	124
pr124	124	59030	59654	1	8	8	1112
bier127	127	118282	120897	2	1	1	274
ch130	130	6110	6336	4	9	9	1165
pr136	136	96772	103069	7	0	0	6
gr137	137	69853	72178	3	107	107	2188
pr144	144	58537	58799	0	0	0	95

* indicates time in milliseconds.

Table A.5: Simple Restart 2

Name	Cities	Best Cost	Cost	%	T_{best}	T_{finish}	Searches
kroA150	150	26524	27790	5	0	0	34
ch150	150	6528	6682	2	578	1143	107097
kroB150	150	26130	27024	3	5	5	543
pr152	152	73682	75518	2	*1254	*1254	93
u159	159	42080	43863	4	0	0	92
si175	175	21407	21518	1	1	1	144
rat195	195	2323	2416	4	1380	2337	134247
d198	198	15780	16124	2	12	12	640
kroB200	200	29437	31130	6	8	8	411
kroA200	200	29368	29912	2	335	1183	56879
gr202	202	40160	42276	5	32	32	286
tsp225	225	3916	4132	5	51	51	2138
ts225	225	126643	129901	3	*1503	*1503	53
pr226	226	80369	81708	2	5	5	195
gr229	229	134602	139276	3	492	963	6138
gil262	262	2378	2484	4	322	1895	55055
pr264	264	49135	52497	7	21	21	531
a280	280	2579	2712	5	608	608	15636
pr299	299	48191	51071	6	210	210	4312
linhp318	318	41345	44170	7	541	541	9503
lin318	318	42029	44170	5	541	541	9503
rd400	400	15281	16235	6	1267	1843	20845
fl417	417	11861	12232	3	3	3	32
gr431	431	171414	181028	6	2750	3600	5839
pr439	439	107217	113675	6	163	993	8585
pcb442	442	50778	54498	7	973	1171	11245
d493	493	35002	36883	5	1199	2816	20063
att532	532	27686	29657	7	725	1575	7397
si535	535	48450	48797	1	123	123	865
ali535	535	202339	217349	7	646	646	605
pa561	561	2763	2990	7	2277	3600	27904
u574	574	36905	39600	7	1332	2186	11061
rat575	575	6773	7328	8	2139	3600	19845
p654	654	34643	35634	3	1178	1178	4038
d657	657	48912	52422	7	991	2176	8065
gr666	666	294358	318619	8	1180	3600	2222

* indicates time in milliseconds.

Table A.6: Simple Restart 3

Name	Cities	Best Cost	Cost	%	T_{best}	T_{finish}	Searches
u724	724	41910	45409	8	1412	3600	11405
rat783	783	8806	9638	9	1079	3600	10010
dsj1000	1000	18659688	21006950	12	616	3600	5039
pr1002	1002	259045	282772	9	1498	3600	5428
si1032	1032	92650	93079	1	1645	3600	5719
u1060	1060	224094	244322	9	946	3600	4663
vm1084	1084	239297	259908	9	1794	3600	4492
pcb1173	1173	56892	63006	11	971	3600	3978
d1291	1291	50801	57508	13	2594	3600	3027
rl1304	1304	252948	277303	10	1424	3461	2695
rl1323	1323	270199	297008	11	1050	3600	2826
fl1400	1400	20127	20952	4	875	1632	1026
fl1577	1577	22249	23932	8	1603	3600	1701
d1655	1655	62128	69920	13	1970	3600	1852
vm1748	1748	336556	370121	10	2596	3601	1532
u1817	1817	57201	66195	16	1706	3600	1610
rl1889	1889	316536	352442	11	2718	3601	1274
d2103	2103	80450	94172	17	2186	3600	1146
u2319	2319	234256	251424	7	2120	3600	1214
pr2392	2392	378032	425330	13	2145	3600	845
fl3795	3795	28772	31487	9	1966	3554	244
fnl4461	4461	182566	203954	12	1900	3605	237
rl5915	5915	565530	651687	15	1177	3616	111
rl5934	5934	556045	643600	16	1643	3616	109
pla7397	7397	23260728	26196014	12	1499	3615	69
rl11849	11849	923288	1074130	16	2338	3662	24
usa13509	13509	19982859	22486516	13	1845	3677	19
brd14051	14051	469385	527657	12	2290	3702	19
d15112	15112	1573084	1767711	12	1779	3687	16
d18512	18512	645238	725477	10	1460	3789	11
pla33810	33810	66048945	77035477	17	2853	4625	2
pla85900	85900	142382641	162793823	14	11418	11418	0

Table A.7: Go With The Winners 1

Name	Cities	Best Cost	Cost	%	T_{best}	T_{finish}	Trasnpos
burma14	14	3323	3323	0	*1	*7	37
ulysses16	16	6859	6859	0	*4	*13	59
gr17	17	2085	2086	0	0	0	7
gr21	21	2707	2707	0	0	*1	32
ulysses22	22	7013	7031	0	*12	*23	64
gr24	24	1272	1309	0	*1	*2	35
fri26	26	937	947	1	0	*3	39
bays29	30	2020	2020	0	*3	*5	51
dantzig42	42	699	708	1	*5	*10	58
swiss42	43	1273	1285	1	*2	*11	56
att48	48	10628	10628	0	*15	*25	50
gr48	48	5046	5124	2	*10	*16	60
hk48	48	11461	11615	1	*4	*9	41
eil51	51	426	433	2	*8	*16	41
berlin52	52	7542	7702	2	*13	*21	55
st70	70	675	705	5	*16	*31	42
pr76	76	108159	110617		0	0	63
eil76	76	538	555	3	0	0	49
gr96	96	55209	56989	3	0	0	69
rat99	99	1211	1256	4	0	0	40
kroE100	100	22068	22887	4	0	0	45
kroC100	100	20749	21435	3	0	0	60
kroD100	100	21294	22319	5	*51	*97	68
kroB100	100	22141	22527	2	0	0	57
kroA100	100	21282	21768	2	0	0	71
rd100	100	7910	8226	4	*39	*79	48
eil101	101	629	658	5	0	0	56
lin105	105	14379	14788	3	0	0	72
pr107	107	44303	44962	1	0	0	54
gr120	120	6942	7216	4	*39	*106	68
pr124	124	59030	59772	1	0	0	63
bier127	127	118282	121859	3	0	0	85
ch130	130	6110	6362	4	0	0	58
pr136	136	96772	104664	8	0	0	47
gr137	137	69853	72567	4	0	1	76
pr144	144	58537	59392	1	0	0	55

* indicates time in milliseconds.

Table A.8: Go With The Winners 2

Name	Cities	Best Cost	Cost	%	T_{best}	T_{finish}	Trasnpos
kroA150	150	26524	27958	5	0	0	58
ch150	150	6528	6692	3	0	0	58
kroB150	150	26130	27101	4	0	0	73
pr152	152	73682	75758	5	*83	*221	64
u159	159	42080	44168	5	0	0	57
si175	175	21407	21535	1	0	0	63
rat195	195	2323	2412	4	0	0	46
d198	198	15780	16141	2	0	0	76
kroB200	200	29437	31207	6	0	0	74
kroA200	200	29368	30014	2	0	0	57
gr202	202	40160	42328	5	1	2	76
tsp225	225	3916	4151	6	0	0	75
ts225	225	126643	131091	4	*190	*339	50
pr226	226	80369	81896	2	0	0	60
gr229	229	134602	139794	4	2	3	77
gil262	262	2378	2480	4	0	0	58
pr264	264	49135	52707	7	0	0	51
a280	280	2579	2727	6	0	0	69
pr299	299	48191	51219	6	0	0	82
linhp318	318	41345	44390	8	0	0	61
lin318	318	42029	44391	7	0	0	61
rd400	400	15281	16199	6	0	1	84
fl417	417	11861	12319	4	0	1	78
gr431	431	171414	179051	4	10	17	87
pr439	439	107217	114047	6	0	2	116
pcb442	442	50778	54640	8	1	1	62
d493	493	35002	36615	3	1	2	88
att532	532	27686	28523	3	1	4	101
si535	535	48450	48812	1	1	1	71
ali535	535	202339	218130	8	7	25	72
pa561	561	2763	2952	7	1	1	67
u574	574	36905	39455	7	1	3	77
rat575	575	6773	7213	7	1	3	77
p654	654	34643	35791	3	1	4	78
d657	657	48912	52500	7	2	5	77
gr666	666	294358	314604	7	30	45	97

* indicates time in milliseconds.

Table A.9: Go With The Winners 3

Name	Cities	Best Cost	Cost	%	T_{best}	T_{finish}	Trasnpos
u724	724	41910	44857	7	3	5	83
rat783	783	8806	9402	7	2	6	79
dsj1000	1000	18659688	20147754	8	16	22	123
pr1002	1002	259045	277869	7	4	11	85
si1032	1032	92650	92952	0	3	5	66
u1060	1060	224094	241859	8	6	14	85
vm1084	1084	239297	254778	6	8	13	81
pcb1173	1173	56892	61930	9	7	15	84
d1291	1291	50801	54172	7	11	21	95
rl1304	1304	252948	275224	9	16	23	96
rl1323	1323	270199	288310	7	13	20	87
fl1400	1400	20127	21012	5	9	35	99
fl1577	1577	22249	23553	6	16	27	70
d1655	1655	62128	66933	7	28	39	113
vm1748	1748	336556	361288	7	35	43	88
u1817	1817	57201	62146	9	25	38	95
rl1889	1889	316536	342537	8	32	54	99
d2103	2103	80450	84152	5	31	52	92
u2319	2319	234256	248120	6	32	48	70
pr2392	2392	378032	410969	8	33	78	94
fl3795	3795	28772	30748	7	166	222	83
fnl4461	4461	182566	195972	7	122	383	155
rl5915	5915	565530	615555	9	418	612	132
rl5934	5934	556045	606518	9	495	703	113
pla7397	7397	23260728	25314898	9	648	1000	114
rl11849	11849	923288	1000858	9	1007	1204	21
usa13509	13509	19982859	21626483	9	1123	1202	7
brd14051	14051	469385	506292	8	1105	1204	7
d15112	15112	1573084	1693679	8	1201	1203	3
d18512	18512	645238	695450	8	1200	1202	0
pla33810	33810	66048945	75337701	14	1199	1200	0
pla85900	85900	142382641	174667635	22	1199	1200	0

BIBLIOGRAPHY

Aldous, D. and U. Vazirani (1994). “Go with the winners”. In: *Foundations of Computer Science, 1994 Proceedings., 35th Annual Symposium on Algorithms*, pp. 492–501. DOI: 10.1109/SFCS.1994.365742.

Croes, G. A. (1958). “A method for solving traveling salesman problems.” In: *Operations Res.* 6, pp. 791–812.

Feo, Thomas A. and Mauricio G.C. Resende (1995). “Greedy Randomized Adaptive Search Procedures”. In: *Journal of Global Optimization* 6, pp. 109–133.

Gabriel Marques Portal, Marcus Rolf Peter Ritt (2012). “An algorithmic study of the machine reassignment problem”. In: *LUME*. URL: http://www.lume.ufrgs.br/handle/10183/54137.

Gutin, Gregory and Abraham P. Punnen, eds. (May 2002). *The Traveling Salesman Problem and Its Variations (Combinatorial Optimization)*. 1st ed. Springer. ISBN: 1402006640. URL: http://www.amazon.com/exec/obidos/redirect?tag=citeulike07-20\&path=ASIN/1402006640.

ROADEF/EURO CHALLENGE (2012). URL: http://challenge.roadef.org/2012/en/index.php.

Saha, Chandan (2013). “Go With the Winners: When more Randomness lowers Chance of Success”. preprint (2013), available at www.mpi-inf.mpg.de/ csaha/gww.pdf.

TSPLIB (Feb. 2013). URL: http://comopt.ifi.uni-heidelberg.de/software/TSPLIB95/.