# HEURISTIC SEARCH METHOD BASED ON BETTING THEORY

**Pedro Demasi**
Federal University of Rio de Janeiro, Informactics Graduate Program
Rua Athos da Silveira Ramos, 274 – CCMN/NCE
Cidade Universitária – Ilha do Fundão – Rio de Janeiro/RJ – Brasil CEP 21941-590
**demasi@ufrj.br**

**Jayme Szwarcfiter**
Federal University of Rio de Janeiro, Informactics Graduate Program
Rua Athos da Silveira Ramos, 274 – CCMN/NCE
Cidade Universitária – Ilha do Fundão – Rio de Janeiro/RJ – Brasil CEP 21941-590
**jayme@nce.ufrj.br**

**Adriano J. O. Cruz**
Federal University of Rio de Janeiro, Informactics Graduate Program
Rua Athos da Silveira Ramos, 274 – CCMN/NCE
Cidade Universitária – Ilha do Fundão – Rio de Janeiro/RJ – Brasil CEP 21941-590
**adriano@nce.ufrj.br**

## ABSTRACT

This work introduces and develops a formal framework for betting theory, proving a simple yet powerful result. It is shown that, given a bookmaker that takes bets on events outcomes and a set of players, it just needs to estimate the behaviors of the players in each outcome instead of the probabilities themselves. A corollary is that a player just needs to be sufficiently (and consistently) better than the average behavior of the pool of players to be successful (i.e. have a consistent positive payoff). Inspired by and based on this, a new search heuristic method is proposed as a practical application, in which the players are actually encoding the problem subspaces estimations and the ones that better estimate the optimal results will survive. The method is applied to the benchmark asymmetrical traveling salesman problem (ATSP). Results are presented and discussed, as well some other possible practical applications.

**KEYWORDS. Metaheuristics. Combinatorial Optimization. Evolutionary Computation.**

**Main area Metaheuristics**

## 1. Introduction

Gambling is an activity that always allured the human being. The act of betting in some event expecting that some outcome will happen may have the effect of validating some opinion (that was so strong that the person felt that it could wager on it) or to protect against some undesirable condition (receiving some profit if something bad happens, as in an insurance policy).

At the same time that gambling may have been mostly considered something associated with people lacking any moral sense, it has also been a field of some interesting mathematical results (Kelly, 1956), (Thorp, 1961). There is also some relationship between stock market, overall investment strategies and gambling (Thorp, 1961), (Thorp and Kassouf, 1967), (Thorp, 1985).

Someone who can earn enough money with gambling, betting on events, to successfully support oneself, might as well as be seen as someone who can infer (or predict) the probabilities of events better than the others. That observation alone seems to carry a great resemblance to evolutionary theories.

If we take evolutionary computation as an example (Jong, 2006), (Michalewicz, 1996), (Michalewicz and Fogel, 2004), the methods deal basically with solutions that are better than the others, that carry some information (or subset of it therefore) that gives them a distinct advantage against the others. Well, that means that if we can, somehow, shape a method in which the "survival of the fittest" is given by how well they can predict the outcomes of given events (which can be seen as subspaces of the search space of a problem in which better solutions are more likely to be found), we could use that information to focus our search on those (smaller) regions of likely good solutions.

This work presents a basic formal betting theory framework which in turn will be the base for a proposed search heuristic method. With that mindset of adapting and mixing evolutionary computation with betting survival, we use the theory described with the new algorithm applied to optimization, comparing results of the benchmark problem asymmetric traveling salesman (ATSP) (Karp, 1972), (Garey and Johnson, 1979), (Reinelt, 1991).

Since it has been proven (Wolpert and Macready, 1997), that no search heuristic method may be equally applied successfully to all different classes of problems (i.e. the average result over all set of problems is constant), it is important to develop and test new methods which can have better results against some classes of problems (or even improve over existing ones).

The next section describes the theoretical framework. The following one proposes the search heuristic method, with the results presented after that. We close with a couple of sections discussing future works and conclusions, respectively.

## 2. A Betting Theory Framework

In this section a basic framework for a betting theory is presented. Some simple concepts and definitions are necessary first, and then an important result will be proven.

### 2.1 Basic Definitions

Let an **outcome** be anything that can happen as a result of some abstract entity that we will call an **event**. Thus, each event has a collection of possible outcomes such as only one (and exact one) of them will actually happen in the future.

In this context, let a **bet** $\beta$ ($\beta \in \mathbb{R}, \beta > 0$) be a wager on any outcome of a given event. When betting on an outcome, the gambler (i.e. the one who placed the bet, which we can simply call **player** from now on) expects to win an amount of resources greater than the amount waged. If the player wages on an outcome that happened, we say that the bet was won, otherwise it was lost.

Each outcome of an event has associated **odds** $\omega$ ($\omega \in \mathbb{R}, \omega > 1$) with it. It means that when placing a bet β on an outcome that has odds $\omega$, the player will receive $\omega\beta$ back if the bet is won for a total profit of $\omega\beta - \beta\omega$. Thus, if the probability of that outcome to happen is $p$ ($p \in \mathbb{R}, 0 \leq p \leq 1$), the expected return $EV$ for the player is given by:

$$\begin{aligned} EV &= p(\omega\beta - \beta) - (1-p)\beta \\ &= p\omega\beta - \beta \\ &= \beta(p\omega - 1) \end{aligned}$$

**Equation 1**

We assume that there is an entity called **bookmaker** (or the **house**), which is responsible for picking events and outcomes, associating odds to outcomes, receiving wagers and paying the winners.

Lets say that $B$ is the set of all bets placed on all outcomes of the set of outcomes $\Omega$ with $b_i$ the sum of bets on outcome $i \in \mathbb{N}, 1 \leq i \leq |\Omega|$ (notice that $|B| = |\Omega|$). Thus, the total amount of bets received by the house is $\sum_{i=1}^{|\Omega|} b_i$. If the outcome that happened is $k \in \mathbb{N}, 1 \leq k \leq |\Omega|$ that means that the house $EV$ is given by:

$$EV = \sum_{i=1}^{|\Omega|} b_i - \omega_k \beta_k$$

**Equation 2**

The total amount of resources that a player has to bet or that the house has to pay the players are both called a **bankroll**. The bankroll changes over time, since a player may win or lose bets, and the house receives bets and pays the winners.

### 2.2 Outcomes and Odds

For every outcome of a given event, the house has to define its associated odds. This is important, since it will make players decide in which outcome to bet (if they decide to bet at all).

To simplify, first we assume that the house is **ideal**, which means it will not make any profit from the bets (of course this is not very realistic, but good enough to develop the definitions we will need).

First, how should an ideal house define the odds for the outcomes of a given event? We suppose that we know for sure the probability $p_i$ of each outcome $i$ happening ($p_i \in \mathbb{R}, 0 \leq p_i \leq 1, \sum_{i=1}^{|\Omega|} p_i = 1$). This is very unrealistic and just for the sake of argument, since we usually have at most some rough estimate.

We know that $\beta_i$ is the sum of bets on outcome $i$ and that $\omega_i\beta_i$ is the total payoff that the house will pay for players should outcome $i$ happens. It means that the expected payoff for each outcome $i$ is $p_i(\omega_i\beta_i - \beta_i)$. So, a different way of writing Equation 2 is:

$$EV = \sum_{i=1}^{|\Omega|} ((1-p_i)\beta_i - p_i(\omega_i\beta_i - \beta_i))$$

**Equation 3**

For an ideal house, we would want $EV = 0$. A simple way to do this is to make the expected payoff zero for each outcome. That leads us to:

$$\begin{aligned} (1-p_i)\beta_i - p_i(\omega_i\beta_i - \beta_i) = 0, 1 \leq i \leq \Omega \\ \beta_i - p_i\omega_i\beta_i = 0 \\ \omega_i = \frac{1}{p_i} \end{aligned}$$

**Equation 4**

Basically Equation 4 tells us that for an ideal house, the probability of an event and its odds are inversely proportional. And it makes sense that the lower the probability of an outcome happening the higher its payoff should be.

The way that a **real** house (i.e. not ideal) could work is to cut a commission from the odds, which is the same as slightly increasing each probability. In this way, the odds could be defined as $\omega_i = \frac{1}{(p_i+\epsilon)}$ for some small $\epsilon$.

This seems good enough, but we usually do not have the precise probabilities of each outcome in a real life event. Having a very a good estimate might not be enough, since in large scale a small error can be a disaster. However, as we will see in the next section, even if the house knew precisely the probabilities and defined a $\epsilon$ as commission, it would not be good enough.

### 2.3 Maximizing Expected Returns

One's expected goal is normally to maximize its own expected return *EV*, which is given by Equation 1 for players and Equation 2 for the house.

To maximize a players *EV*, from Equation 1 we know that it should be the case that $p\omega > 1$. It is also clear that, since $0 \leq p \leq 1$ there would be no point in betting if $\omega \leq 1$. Thus, if Equation 4 holds, the player's *EV* will always be zero.

For the house, however, it might not be a good idea to follow Equation 4, if it is possible to know with all precision the probabilities of the outcomes happening. If we suppose that the house's bankroll is not infinite, and that the sum of bankrolls of a large amount of players may account for a good ration of the house's bankroll, then a short streak of "bad luck" may actually bankrupt the house even if the expected returns are positive in the long run.

Suppose that there is some improbable (i.e. low probability, approaching zero) outcome. In this case, since $\omega = \frac{1}{p}$, that means that $\omega$ will be really large. If many players decide to bet on this outcome hoping for a big win and it turns out to happen, the house might be in big trouble.

So, it is not enough to make sure that your expected return, in the long run, is positive. The house has to make sure that it has a positive expected return for every event.

Looking again at Equation 2 we can conclude that the house will break even (i.e. not profit nor loss) if, and only if, for every *k*:

$$\omega_k = \frac{\sum_{i=1}^{|\Omega|} b_i}{\beta_k}$$
$$\frac{1}{\omega_k} = \frac{\beta_k}{\sum_{i=1}^{|\Omega|} b_i}$$

**Equation 5**

What Equation 5 means is that, instead of being inversely proportional to the probability of an outcome, the odds must actually be inversely proportional to the ratio of bets made on the outcome. So, by bringing the commission $\epsilon$ into account, the house will be certain to have a positive expected return for all outcomes of every event.

This is a simple yet powerful result. It is basically saying that it is better to know the probability that each player will bet in each outcome than to know the actual probabilities of the outcomes. It does not even matter the event or the outcomes if the house knows how players will behave. In short, **knowing the players behavior is better than knowing the event outcome chances**.

If we take in consideration the efficient market hypothesis (Jones and Netter, 2008) it would mean that the ratio of bets on any outcome could be interpreted as close to the probability of that outcome.

By this result, the odds of an event set by the house will be based in the player's behaviors. Thus, a corollary of this result is that in order to have a positive return in the long run, a player has to consistently infer the odds of the events better than the other players. This means that a player does not have to predict correctly (as neither does the house), it just has to be better than the pool of all players (even though this might not look as a direct competition in the first

place).

### 2.4 Discussion

One of the interesting consequences of last section results is that it is perfectly possible for a player to have a positive *EV*, even if the house will always have a positive *EV* too.

If we interpret the ratios of bets in each outcome (which in turn decide the odds) as the estimates of the group of players for each outcome, then it basically means that, as briefly discussed, in order to have $EV > 0$, a player must be able to estimate the probabilities of the outcomes better than all of the players as a group.

It is also the case that the house has to, somehow, estimate the ratio of bets for each outcome in order to define the odds. While this is true, it is always possible for the house to adjust the odds if its estimates turn out to be a little off.

So, assuming that the house will keep its positive *EV*, it means that its bankroll will always grow. This is not, however, true for all players. Supposing that there are no external sources of income (i.e. a player's bankroll only grows if it wins a bet), a negative *EV* player will eventually be bankrupt.

## 3. Application

In the last section a framework for betting theory was defined and described with an important result. This section aims to take advantage of all of this to derive a search heuristic method.

As discussed in 2.4, it is perfectly possible that a player (or even more than one) and the house have a positive *EV*. In this context, it seems that a player with a positive *EV* is basically *better*, in some sense, than the other players.

If we think of players as algorithms, we can interpret this as competing algorithms and the eventually bankrupt ones are removed from the pool of players and possibly replaced with new algorithms.

This is a very high level and broad definition of how what we have described so far may be applied on a more practical level. We still have to decide what the algorithms (players) actually mean, what are the bets they are taking, what do the bankrolls represent, and who is the house etc.

### 3.1 A Betting Game

Let $f: D \to \mathbb{R}, D \subset \mathbb{N}$ be a function for which we want to find $x^* \in D$ such as $f(x^*) \geq f(x), \forall x \in D$. We are assuming that $f(x)$ can be evaluated in polynomial time, given $x$. At every step of the method, we will have a current best-known candidate solution $x^*$.

The first thing we have to define is how we represent each candidate solution. Since $x \in D$ is a natural number, each candidate solution could be represented by a string of $N$ bits, which means the total search space for our function would be $2^N$. That does not prevent, of course, other ways of representation, like an array of integers that could possibly represent a permutation of nodes of a graph, for instance. Either way, we will use $N$ to represent the size of our candidate solutions, which means the number of elements in this set.

The next important concept is the *mask* $M$, which is a set of size $|M|$. Every step of our method will deal with the possibility of making changes to the current best-known solution by applying this mask. Because of this, the mask can be either of the same encoding as the candidate solution but can also be of something else, for instance a list of unique indices.

There will also be $m$ different applications of the mask, which directly translates to all possible outcomes of an event (this will be what the players will bet on). So, if we think that each element in $M$ may or may not be applied to the current best solution $x^*$, that would mean that $m = 2^{|M|}$. It could also be the case that we just apply each element of $M$ to $x^*$, so in that case we would have $m = |M|$.

Overall, we can think of the mask as a transformation $\tau_i: D \to D, 0 \leq i \leq m$, in which each $\tau_i$ is defined based on $M$, and we apply $\tau_i(x^*), 0 \leq i \leq m$ to create new candidate solutions that represent the possible outcomes of an event.

For instance, suppose that we have a string of bits representing each candidate solution, that $M$ is a set of the indices of the string (thus$0 \leq M_i < N$), with $m = 2^{|M|}$ and that each $\tau_i(x^*)$ is defined as flipping the bit $M_j$ of $x^*$ if, and only if, the $M_j$-th bit of $i$ is set. In this case, suppose that $N = 200$ and $M = \{57, 123, 190\}$, then for $\tau_0$ we would not flip any bits, for $\tau_2$ we would flip bit 123 and for $\tau_5$ we would flip bits 57 and 190.

Each one of the $m$ applications of the mask will be an outcome, so each outcome is defined by $\tau_i$. The winner outcome $\tau_j$ will be the one such that $\tau_j > \tau_i, 0 \leq i < m, i \neq j$.

Each player will consist of an algorithm or formula to decide the weight of a given $\tau_i$. That means that each player should be able to, given mask $M$, and knowing the definition of $\tau$, to calculate the weight $w$ of each outcome. What this basically means is that there will be $m$ candidate solutions (possible outcomes) that will be creating by applying a transformation in the current known best, one of those will be the best one (winner). Given the calculated weights $w_1, \ldots, w_m$ for every combination of $m$, the estimated probability $p'$ for each outcome is:

$$p'_k = \frac{w_k}{\sum_{i=1}^{m} w_i}$$

**Equation 6**

The way to calculate the weights can vary and is probably one of the most important definitions for applying the method to find a solution to a problem. So, for instance, we will keep using the example definition of $\tau_i$ we did before. One simple way of defining the weight calculation for each player would be to have an array $p1$ (with $|p1| = N$ of real numbers that define probability that this particular bit being changed is good. It means that, given $M$ and $m$, and considering $M_j$ for all $j$ bits set in $i$ and $M_k$ for all bits not set in $i$, we would have $w_i = \left(\prod p1_{M_j}\right) \times \left(\prod 1 - p1_{M_k}\right)$.

So, for instance, for the example above with $\tau_5$ (bits 0 and 2 set, 1 unset with $M = \{57, 123, 190\}$) we would have $w_5 = p1_{57} \times p1_{190} \times (1 - p1_{123})$.

Each player will, therefore, calculate its estimated probabilities for each outcome and then look for all the odds for the one that maximizes the value of Equation 1, which is $p'_k \omega_k$.

The size of the bet $\beta$ will follow the Kelly Criterion (Kelly, 1956), which is a fraction of the current bankroll $\rho$ of the player and is given by:

$$\frac{\beta}{\rho} = \frac{p'(\omega - 1) - (1 - p')}{\omega - 1}$$
$$\beta = \frac{p'\omega - 1}{\omega - 1}\rho$$

**Equation 7**

If Equation 7 yields a result lesser than some predetermined minimum $\beta_{min}$, then that minimum should be used instead, unless it is greater than the bankroll. So, the actual value of $\beta$ is:

$$\beta = \min\left(\rho, max\left(\beta_{min}, \frac{p'\omega - 1}{\omega - 1}\rho\right)\right)$$

**Equation 8**

The size of the player pool is given by $P$. Each player $\pi$ will start with a bankroll of $\rho_\pi$. We assume that when a player goes bankrupt, another one will be summoned to take its place. The new player can be a completely random one (with regards to its weight calculation) or it could be created based on the players with currently top bankroll (applying any transformation or delta change on its values), or even a combination of both (with probabilities for each one).

### 3.2 The Method

The method is divided in two stages (second being optional). In the first stage, we have a number of arbitrary iterations (or any kind of stopping condiation may apply). Each iteration is an event consisting of $m$ outcomes, with the mask $M$ created randomly. As mentioned before, a known best solution $f(x^*)$ is kept (and used as base to apply the masks), so for the first step a completely random solution may be used as $x^*$ (if desired, some known *good* solution may be used). In each step, initially $\omega_i = \frac{1}{m}, \forall i$. Another way to possibly solve this is to precalculate all players weights (since they will not change anyway) and have all $\omega_i$ estimated based on that (i.e. the average probability in each one), which might also require some adjustments.

There is a player pool from which players are randomly picked to bet (actually a simple linear shuffle may be performed on an array consisting of players indices).

Each player then calculates the estimated probabilities for each outcome (based on its own rules and formulas) and selects the outcome $k$ which gives the maximum $EV$ (tie breaking rules may apply). The size of the bet $\beta$ is given by Equation 8.

After each player bets, the values for all odds $\omega$ may be recalculated to adjust to any trend, since we discussed before that the house have to adjust its estimates based on how the players behave.

When all players have bet, each outcome $m$ is evaluated applying $f(.)$and the one with the overall best solution is the winner. The players which bet on the winner outcome receive their respective payoffs. If there are any players bankrupt, they are removed from the player pool and new players are summoned to take their place. As discussed, new players may be created either by randomizing each bit or by adjusting the current best players (with bigger bankrolls) by adding some random value $-\epsilon < x < \epsilon$ to each weight. If the winning solution have a better $f(.)$than $f(x^*)$ then update $x^*$ accordingly.

After the adjustments are done, a new iteration is run, with new masks. When a certain number of iterations were run (or some other criteria is met), the first stage is over.

The second stage is run with a given (possibly small) number of the top players from the first stage. In this stage, candidate solutions may be created given the best players weight calculations randomized. For instance, with the examples discussed before, the $p1$ encoding could be seen as probabilities and randomly selecting each one. This means that, for each $p1_i$ for a player, set that bit as 1 with $p1_i$ probability. The reasoning behind this approach is that each player has a well-adjusted probability for each bit, and using it directly (in a randomized manner) tends to yield the best results. This can be run for some fixed amount of time or until some other arbitrary criteria are met.

The reasoning behind this step is that after a (sufficient) large number of iterations in the first stage, the players who best calculate the weights of the masks are the ones to survive and, thus, they are the best "pickers".

After a predetermined number of iterations (or any other criteria, like time elapsed), the stage is over and the method reports the best solution it founded.

### 3.3 Algorithmic Description

The algorithms below formulate the method described in previous sections. We are using the same variables and arguments and they have been used throughout this text. In that way $\omega$ refers to odds, $M$ is the mask (with $|M|$ its size), $m$ is the total number of masks permutations, $\tau$ is the transformation applied to the best solution, $N$ is the size of candidate solutions, $\beta$ is a bet, $\rho$ is the bankroll, $w$ is the weight of a given mask permutation calculated by the player, $p1$ is the probability of a given bit to be set (1), $P$ is the player pool (with size $|P|$), $x^*$ is the current best known solution and $f(.)$is the function being optimized. We also assumed some constant arguments are also set (like the number of iterations of the method, size of player pool etc).

One thing to keep in mind is that, while the pseudo-code below assumes that we want to maximize function $f(.)$, it might be the case that we want to minimize it and, of course, adjusments should be done accordingly.

Algorithm 1 describes the first stage of the method, while Algorithm 2 describes one possible application of a second stage. Also, Algorithm 3 describes a way to create a new player based on some other player (instead of plainly random).

Pick random candidate solution and assign to $x*$
**for** each iteration **do**
  Select random mask $M$ of size $|M|$
  **for** each player $i$ in player pool $P$ **do**
    **for** j from 1 to $m$ **do**
      Calculate weight $w_{i,j}$
    **end for**
  **end for**
  **for** j from 1 to $m$ **do**
    $\omega_j \leftarrow \frac{\sum_{i=1}^{|P|} w_{i,j}}{\sum_{k=1}^{m}\sum_{i=1}^{|P|} w_{i,k}}$
  **end for**
  **for** each player $i$ in player pool $P$ **do**
    **for all** $k$ such that $\omega_k > w_{i,k}$ **do**
      $p' \leftarrow \frac{1}{w_{i,k}}$
      $\omega \leftarrow \omega_k$
      $\beta_{i,k} \leftarrow min(\rho, max(\beta_{min}, \frac{p'\omega-1}{\omega-1}\rho))$
      bet $\beta_{i,k}$ in outcome $k$
    **end for**
  **end for**
  $best \leftarrow 0$
  **for** each mask $i$ in $m$ **do**
    $t \leftarrow f(\tau_i)$
    **if** $t > best$ **then**
      $best \leftarrow t$
      $b \leftarrow i$
    **end if**
  **end for**
  **for** each player $i$ in $P$ **do**
    **if** player bet on $b$ **then**
      $\rho_i = \rho_i + \beta_{i,b} \times \omega_b$
    **end if**
    **if** $\rho = 0$ **then**
      Remove player from pool
      Create and insert new player in pool
    **end if**
  **end for**
  **if** $best > f(x*)$ **then**
    $x* \leftarrow \tau_b$
  **end if**
**end for**

**Algorithm 1**

```
for each iteration do
  for each player do
    Create solution s ← 0
    for each bit 0 < i ≤ N do
      Set bit i of s with probability p1_i
    end for
    Evaluate f(s)
  end for
end for
```

**Algorithm 2**

```
Require: 0 < ε < 1
  Select player π from current pool
  w' ← w_π
  for each bit 0 < i ≤ N do
    θ ← rand(−ε, ε)
    w_i ← w'_i + θ
    wi = min(1, w_i)
    wi = max(0, w_i)
  end for
```

**Algorithm 3**

### 3.4 Parameters

As with all search heuristics (Michalewicz and Fogel, 2004), there are a few parameters that should be set to run this method, and fine tuning them might yield better results.

Given the function $f(.)$we are trying to optimize, the search space size should be defined by each candidate solution encoding size.

For each iteration of the method, we should define a mask $M$ (which can be random or obeying some kind of criteria). The size $|M|$ of the mask can vary on each iteration or stay the same.

The amount of candidate solutions evaluation executed for each mask combination $m$ in order to verify the winning outcome should also be determined. It could be a function of the mask size or just constant, but it should be the same for every outcome in the same event.

The size of the player pool $P$, each player's bankroll $\rho$ and the minimum bet $\beta_{min}$ should also be predetermined a possibly stay constant throughout the execution of the method.

Finally, the amount of iterations in each stage of the method could also be predetermined (or complying to some external criteria like execution time).

## 4. ATSP

In this section we discuss the results of applying the proposed method on the asymmetric traveling salesman problem (ATSP). Given a directed graph $G = \{V, E\}$ and a distance matrix $D = (d_{ij})$ of dimension $n \times n$, the ATSP can be defined as calculate a closed circuit (which can be represented by a any permutation $P$ of the set of vertices $\{1, \dots, |V|\}$) that minimizes

$$d_{S_n S_1} + \sum_{i=1}^{n-1} d_{S_i S_{i+1}}$$

**Equation 9**

If we had $d_{ij} = d_{ji}, \forall i, j$, then that would be the symmetric variation of TSP, but in this case there can be cases where this does not hold.

The TSP is a good example of an NP-hard problem (Karp, 1972), (Garey and Johnson, 1979), (Cormen et al, 2009) and thus very suitable as a benchmark to application of optimization techniques and search heuristic methods. The examples used here are from the well-known **TSPLIB** (Reinelt, 1991) that are broadly available and used in benchmark experiments.

### 4.1 Setup

We applied the method to some instances of the ATSP found in (Reinelt, 1991). There were 20 independent runs for each instance, with the average, minimum and maximum results recorded and shown on Table 1.

Each candidate solution was encoded as an array of integers, being a simple permutation of the set $\{1, \dots, |V|\}$, which means that there are no repeated elements.

A mask of size 3 was used, and the application of the mask meant that if $M_i$ was used then 5 elements are removed from the candidate solution (with indices $M_i, M_{i+1}, \dots, M_{i+4}$) and reinserted greedly (which means reinserting them in the permutation in the way that minimizes the weight increase). We have to note that $M_i \leq |V| - 4$. Given that each element of the mask might or might not be used to create a new solution we have $m = 2^{|M|} = 8$ possible outcomes.

The weight calculation for players follow the simple rules already exemplified before. That means that each player has an array of real numbers stating the probability that the particular mask index should be set.

There was used a pool of 25 players and 200,000 iterations of the method. The bankroll was defined as 1000, the minimum bet as 1 and there was a maximum bet of 200. The new players created after a removal of the pool were done 50% of the time as completely random and 50% using Algorithm 3.

### 4.2 Results and Discussion

Results are given on Table 1. Each instance is referred by the name as in (Reinelt, 1991), with the respective $|V|$ size and optimal values. We also represented the average value of the 20 runs, as well as the best and worst results. The percentage column is given by the ratio of the average and the optimal in excess. All runs were done in less than 10 seconds in a 2012 MacBook Pro 2.3 GHz and 8 Gb RAM. The code was done all in **C++** and compiled using *gcc*.

| Instance | \|V\| | average | best | worst | original | % |
|---|---|---|---|---|---|---|
| br17 | 17 | 39.0 | 39 | 39 | 39 | 0.00 |
| ftv33 | 34 | 1355.0 | 1298 | 1392 | 1286 | 5.37 |
| ft53 | 53 | 7437.0 | 7017 | 7907 | 6905 | 7.71 |
| ft70 | 70 | 40416.1 | 39853 | 40874 | 38673 | 4.51 |
| kro124 | 100 | 40875.4 | 38075 | 44159 | 36230 | 12.82 |
| rbg323 | 323 | 1521.00 | 1484 | 1560 | 1326 | 14.71 |

**Table 1**

The proposed method seems promising. It is based on novel criteria and it may suit problems of interest. Although it looks more suitable for dynamic problems it still does a nice job of finding fast good solutions for hard problems with huge search spaces, as shown by the results, which we can also compare with some other works and results such as (Glover et al, 2000), (Sun et al, 2005) (Nagata and Soler, 2012). We can see that the current implementation of the method performs better than some implementations of the cited works, and even than some heuristics.

## 5. Conclusion

This work introduced a basic framework for a betting theory, showing the important result that it is better to know what your opponents are likely to do (i.e. bet in different outcomes) than to actually know the probabilities of the outcomes happening.

That result and the theoretical framework presented inspired a new search heuristic method. This new method was applied on benchmark optimization problem asymmetric traveling salesman (ATSP) to test its real potential.

While there is still much left to do and improve, as illustrated last section, this work presented some novel basic theory and a new optimization technique that can be really usefull in tackling different hard problems.

The results also show some great potential, and this new approach might prove to be

very useful for different classes of problems in the future.

## 6. Future Works

There are still some improvements left to do, either on the theoretical part as well as in the application part.

The betting theory framework can be expanded to make it more generic and accommodate some different forms of betting and wagering.

On the optimization field, application of the method to a broader and different benchmark problems, as well as unique and different ones, possibly real-time optimization. There is also the need to apply the method to classes of problems in which evolutionary computation (with methods like genetic algorithms) are known to perform poorly, which given the "No Free Lunch" theorem (Wolpert and Macready, 1997) may be a good investigation.

Some tweaks and improvements on the method may prove to be very useful too, such as testing hybrid algorithms and methods to verify in which cases better results can be achieved. One of the paths that might yield good results is to combine the method with genetic algorithms, using the betting as the environment for solvers of problems (thus, surviving the best solvers that can estimate better how to deal with different classes of problems). This is a little more complex approach, but definitely worth investigating.

Trying better and more complex weight calculation for the masks might be also very important in order to achieve better results. Better calculations for the odds in each iteration of the method may also prove useful.

Some interesting further work that may be done with the proposed method is to reuse players from different instances (i.e. different functions) but in the same class of problems (e.g. different instances of the traveling salesman problem).

There is also some interesting work that can be done with parallelization. Any given number of top $k$ players chosen for the second stage may be parallelized. It is also possible to parallelize the weight calculation for all players in the first stage (although is not possible to parallelize the odds and bet size calculations, since those rely in serial betting).

## References

**Cormen, T. H., Leiserson, C. E., Rivest, R. L. and Stein, C.**, *Introduction to Algorithms*. The MIT Press, 2009.

**Garey, M. R. and Johnson, D. S.**, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, 1979.

**Glover, F., Gutin, G., Yeo, A. and Zverovich, A.**, *Construction heuristics for the asymmetric tsp*, European Journal of Operational Research, 2000.

**Jones, S. L. and Netter, J. M.**, *Efficient capital markets*, Concise Encyclopedia of Economics, Indianapolis: Library of Economics and Liberty, 2008.

**Jong, K. A. D.**, *Evolutionary computation: a unified approach*, MIT Press, 2006.

**Karp, R. M.**, *Reducibility among combinatorial problems*, Complexity of Computer Computations, pp 85-103, 1972.

**Kelly, J. L.**, *A new interpretation of information rate*, Bell System Technical Journal, 35:917-926, 1956.

**Michalewicz, Z.**, *Genetic Algorithms + Data Structures = Evolution Programs*, Springer, 1996.

**Michalewicz, Z. and Fogel, D.**, *How to Solve It: Modern Heuristics*, Springer, 2004.

**Nagata, Y. and Soler, D.**, *A new genetic algorithm for the asymmetric traveling salesman problem*, Expert Systems with Applications, 2012.

**Reinelt, G.**, *Tsplib - a traveling salesman problem library*, European Journal of Operations Research, 52(125), 1991.

**Sun, J., Zhang, Q. and Tsang, E.**, *A new evolutionary algorithm for global optimization*, Information Sciences, (169):249--262, 2005.

**Thorp, E. O.**, *Fortune's formula: The game of blackjack*, American Mathematical Society, 1961.

**Thorp, E. O.**, *The Mathematics of Gambling*, Lyle Stuart, 1985.

**Thorp, E. O. and Kassouf, S. T.**, *Beat the Market: A Scientific Stock Market System*, Random House, 1967.
**Wolpert, D. H. and Macready, W. G.**, *No free lunch theorems for optimization*, IEEE Transactions on Evolutionary Computation, 1(67), 1997.