Capitolo 1

Analisi Sperimentale

1.1 La libreria TPTP

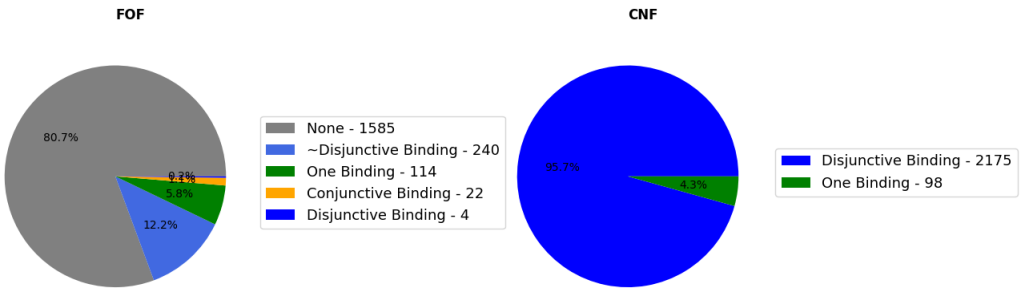

Figura 1.1: Classificazione Libreria TPTP fof e cnf senza uguaglianza

Per verificare la correttezza e l'efficienza dell'algoritmo implementato è stata scelta la libreria di problemi TPTP [**?**] come dataset di problemi da risolvere. La libreria TPTP è una collezione di problemi per sistemi ATP dove ogni problema è scritto in formato TPTP come descritto nella sezione **??**. I problemi TPTP sono suddivisi in base al dominio di appartenenza e il formato (fof, cnf, tff, ecc). Il nome dei file segue il seguente schema:

```
<domain><number>[+,-]{.<variant>}.p
```

dove '<domain>' è il dominio di appartenenza del problema composto da tre lettere maiuscole e '<number>' è un numero, generalmente di tre cifre che identifica il problema all'interno del dominio. Il simbolo + indica che il problema è scritto con la sintassi *fof* mentre il simbolo - indica che il problema è scritto con la sintassi *cnf*. '.<variant>' è un suffisso opzionale che identifica una variante del problema ed in generale è un numero a tre cifre. Ad esempio un nome valido è *SYN001+1.003.p* che indica il problema 1 del dominio *Syntactic*, in formato *fof*, variante 3.

Non tutti i problemi sono adatti per essere risolti con l'algoritmo implementato ed è stato quindi necessario filtrare i problemi in base a determinate caratteristiche. In primo luogo sono stati scartati tutti i problemi non in formato *fof* o *cnf* e i problemi con uguaglianza. Questo è stato possibile tramite il comando *TPTP2T*:

- Per i problemi fof: `tptp2T -q2 -pps Form FOF -Equality`
- Per i problemi cnf: `tptp2T -q2 -pps Form CNF -Equality`

Il risultato delle due query ha restituito una lista di **1969** problemi in formato *fof* e **2274** problemi in formato *cnf*. Da entrambe le liste è stato scartato un problema puramente proposizionale, quindi poco significativo per la sperimentazione, ma estremamente grande da rallentare l'intero set di benchmark, riducendo il numero di potenziali problemi utili a **1965** per i problemi *fof* e **2273** per i problemi *cnf*.

Successivamente tutti i problemi sono stati classificati tramite il classificatore descritto nella sezione **??**. Si ricorda che i problemi sono dati in formato $(A_1 \wedge A_2 \wedge ... \wedge A_n) \Rightarrow C$ dove $A_1, A_2, ..., A_n$ sono assiomi e C è la congettura, mentre sia l'algoritmo di decisione che Vampire lavorano sul problema negato ovvero $A_1 \wedge A_2 \wedge ... \wedge A_n \wedge \neg C$. Per classificazione di un problema P con congettura si intende quindi il frammento di appartenenza del problema negato $\neg P$. Nelle formule in cui non è presente la congettura, come quelle del formato *cnf*, per classificazione della formula P si intende il frammento di appartenenza del problema non negato $P = A_1 \wedge ... \wedge A_n$. I risultati della classificazione sono mostrati in figura 1.1.

Riguardo i problemi *fof*, dei 1965 problemi analizzati:

- **1585** sono stati classificati come *None* e quindi totalmente inutilizzabili.
- **114** sono *One Binding* e **22** sono *Conjunctive Binding*, quindi adatti per l'algoritmo.
- **244** sono *Disjunctive Binding* e quindi non adatti per l'algoritmo.

Dei 244 problemi *Disjunctive Binding*: **240** contengono la congettura mentre gli altri **4** non la contengono. I 240 problemi con la congettura sono stati recuperati negandoli in modo tale da portarli nel formato $\neg A_1 \vee ... \vee \neg A_n \vee C$ che fa parte del frammento *Conjunctive Binding* e quindi adatto per l'algoritmo (questi problemi nella figura 1.1 sono chiamati $\sim$*Disjunctive Binding*). Questo porta ad un numero totale di **376** problemi utili in formato *fof*.

Riguardo i problemi *cnf*, la suddivisione è più netta. Tutte le formule CNF sono infatti o del frammento *One Binding*, se per ogni clausola tutti i letterali della clausola hanno la stessa lista di termini, o altrimenti del frammento *Disjunctive Binding*. Dei 2273 problemi analizzati:

- **98** sono del frammento *One Binding*
- **2175** sono del frammento *Disjunctive Binding*

In questo caso la negazione delle formule *Disjunctive Binding* porterebbe a problemi puramente proposizionali e quindi poco utili per la sperimentazione. Il totale dei problemi utili in formato *cnf* è quindi di **98**. Per un totale di **474** problemi utili. La lista dei 474 problemi è riportata nell'appendice **??**. Ogni problema è stato numerato nella tabella **??** per essere facilmente identificato.

1.2 Analisi dei risultati

In questa sezione verranno analizzati i risultati dell'esecuzione dell'algoritmo sui problemi selezionati nel paragrafo precedente e confrontati con i risultati ottenuti da Vampire. L'obbiettivo della sperimentazione è confrontare l'efficienza di un algoritmo gneral purpose basato su Resolution come quello implementato da Vampire con un algoritmo specializzato SMT come quello implementato. Vampire implementa numerose strategie, euristiche e inferenze di semplificazione per essere efficiente a livello competitivo quindi per indurlo a comportarsi il più possibile come il modello della *Given Clause* descritta nella sezione **??** è stato necessario disabilitare/impostare alcune opzioni. In particolare sono state disattivate tutte le semplificazioni e le euristiche applicabili ad entrambi gli approcci ma non presenti in modo comune alle attuali implementazioni. L'algoritmo di saturazione adottato è stato *Otter*, poiché l'algoritmo predefinito *LRS* non offre garanzie di completezza e si basa sull'uso di limiti

di tempo e memoria come criteri di selezione/semplificazione. È preferibile evitare questa metodologia poiché si desidera che gli algoritmi confrontati abbiano quantomeno lo stesso input. Anche per il preprocessing sono state disabilitate tutte le semplificazioni non comuni a entrambi gli algoritmi. In particolare l'opzione *-updr* (Unused Predicate Definition Removal) è stata disabilitata in quanto non utilizzata dall'algoritmo Binding. Come regola di semplificazione è stata disattivata l'opzione *-fs* (Forward Subsumption) che elimina clausole che sono sussunte da altre clausole durante la fase di *Forward simplification* (Una clausola D è sussunta da una clausola C se esiste una sostituzione σ tale che $C^\sigma \subseteq D$, clausole del genere vengono cercate dalla fs e rimosse perchè ridondanti). È stata disattivata anche l'opzione *-av* (AVATAR - Advanced Vampire Architecture for Theories and Resolution) che è un metodo SMT implementato in Vampire per lo splitting delle clausole. L'opzione *-av* è stata disattivata dato che non si vuole che Vampire utilizzi metodi SMT per la risoluzione dei problemi. Il comando utilizzato per l'esecuzione di Vampire è quindi il seguente:

```
vampire --mode vampire -sa otter -t 10m -m 12000 -av off -updr off -fs off <problem>
```

Dove *<problem>* è il problema da risolvere. Come limiti di tempo e memoria sono stati impostati rispettivamente 10 minuti e 12GB di ram. L'algoritmo Binding è stato eseguito con i seguenti parametri:

```
vampire --mode 1b -t 10m -m 12000 <problem>
```

I seguenti risultati sono estrapolati dall'esecuzione del programma su un Macbook Pro 2018, 2.9 GHz 6-Core Intel Core i9, 16 GB 2400 MHz DDR4 sistema operativo macOS Sonoma 14.0. Gli esperimenti sono stati poi ripetuti su un computer Windows 11 con processore Intel Core i9-13900K e ram 32GB DDR5 sul sottosistema Windows for Linux (WSL) e si sono ottenuti tempi di esecuzione, come da aspettativa, più bassi ma assolutamente coerenti, mentre per memoria i valori sono rimasti esattamente gli stessi. I tempi delle tabelle si riferiscono ai tempi rilevati dalla macro $Time\_Trace$ posta all'inizio dell'algoritmo Binding e all'inizio del *MainLoop* di Vampire escludendo quindi i tempi di parsing e preprocessing. Per una maggior accuratezza, i tempi sono stati calcolati come media di 5 esecuzioni.

One Binding

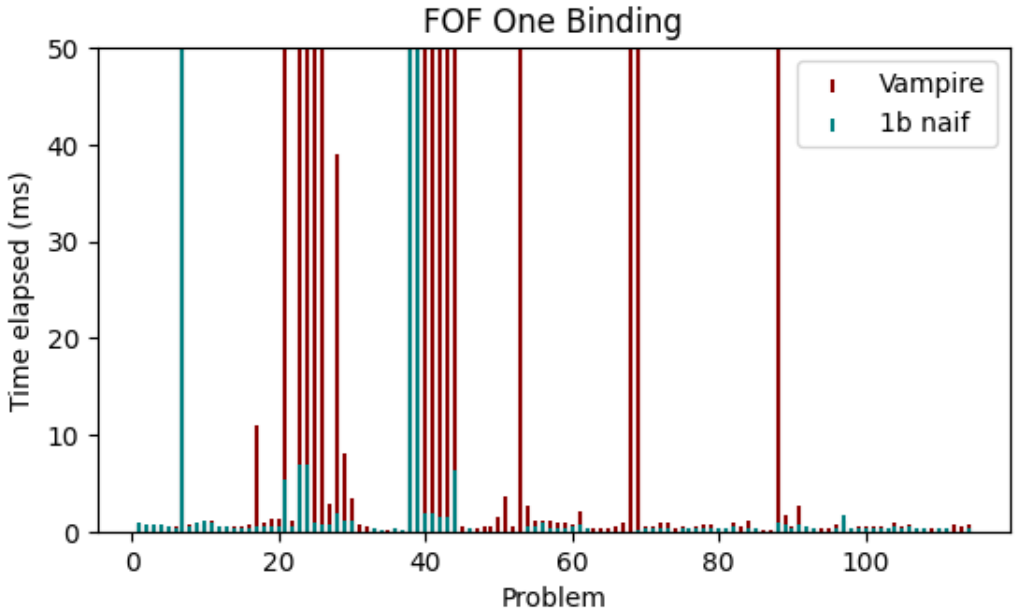

Figura 1.2: Tempo Vampire vs 1b naif, problemi fof

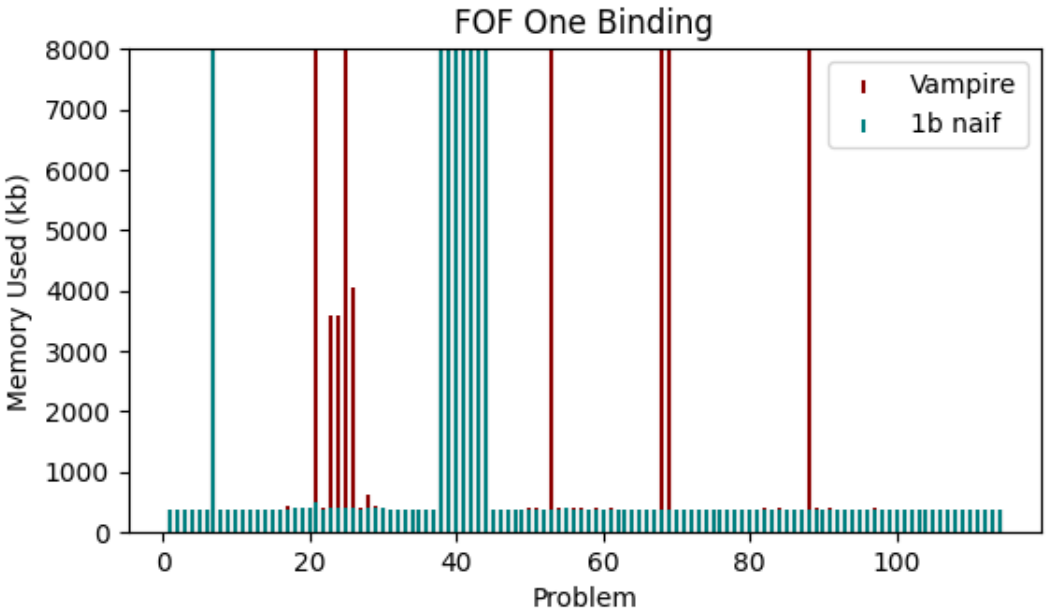

Figura 1.3: Memoria Vampire vs 1b naif, problemi fof

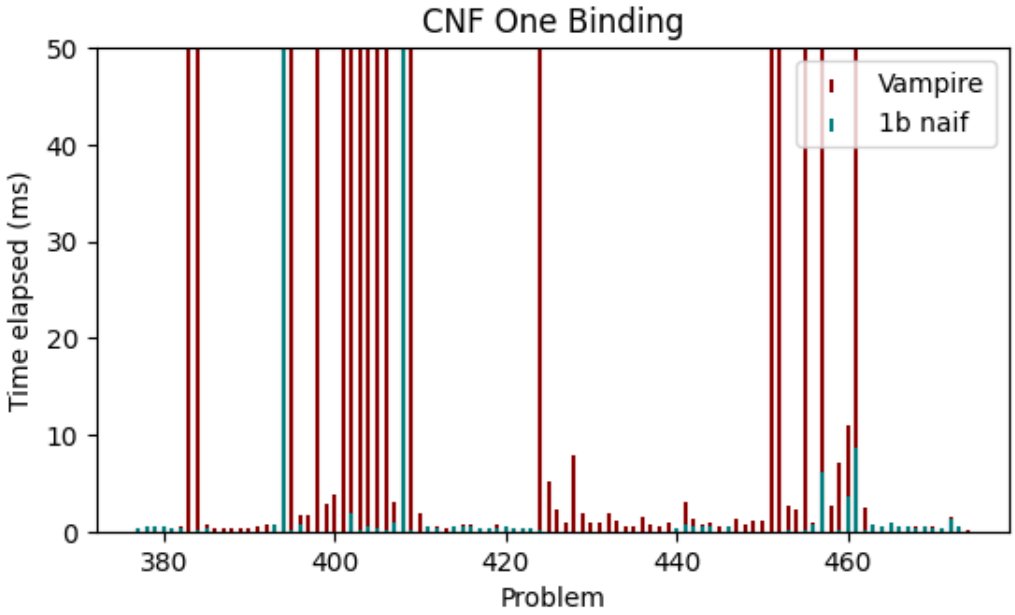

Figura 1.4: Tempo Vampire vs 1b naif, problemi cnf

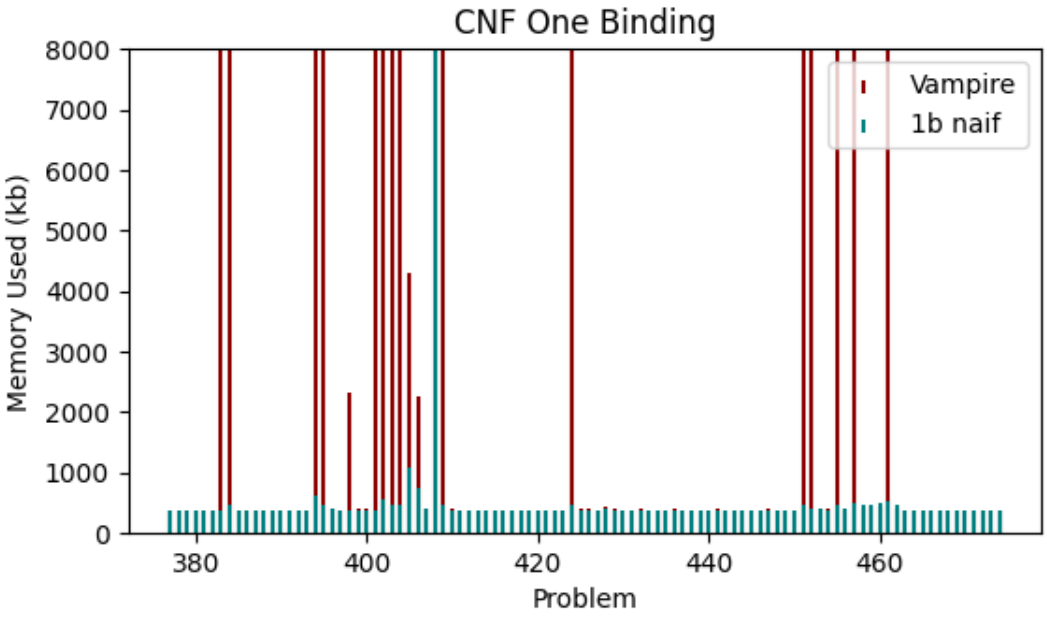

Figura 1.5: Memoria Vampire vs 1b naif, problemi cnf

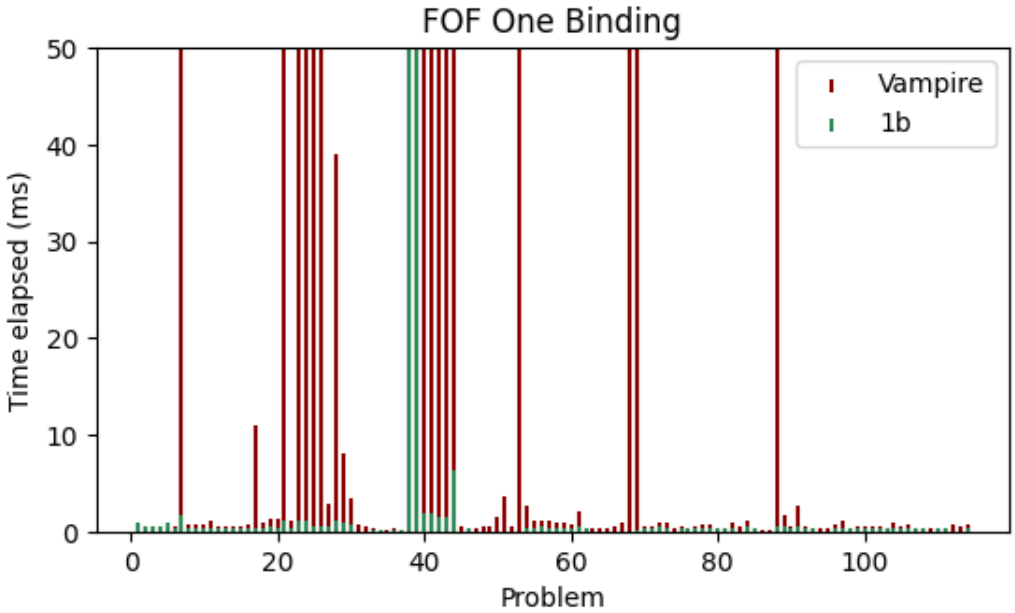

Figura 1.6: Tempo Vampire vs 1b, problemi fof

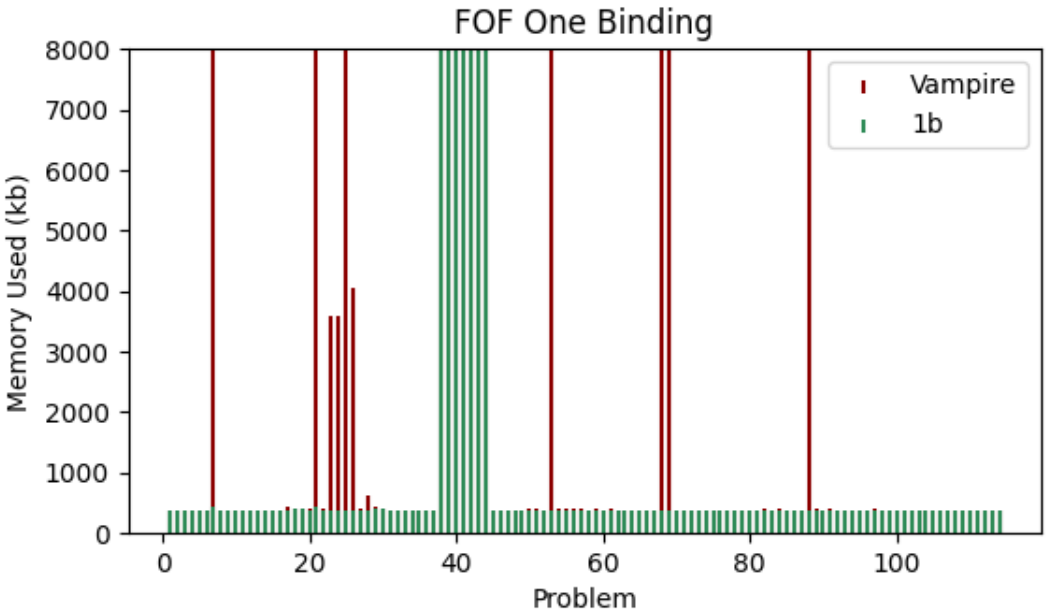

Figura 1.7: Memoria Vampire vs 1b, problemi fof

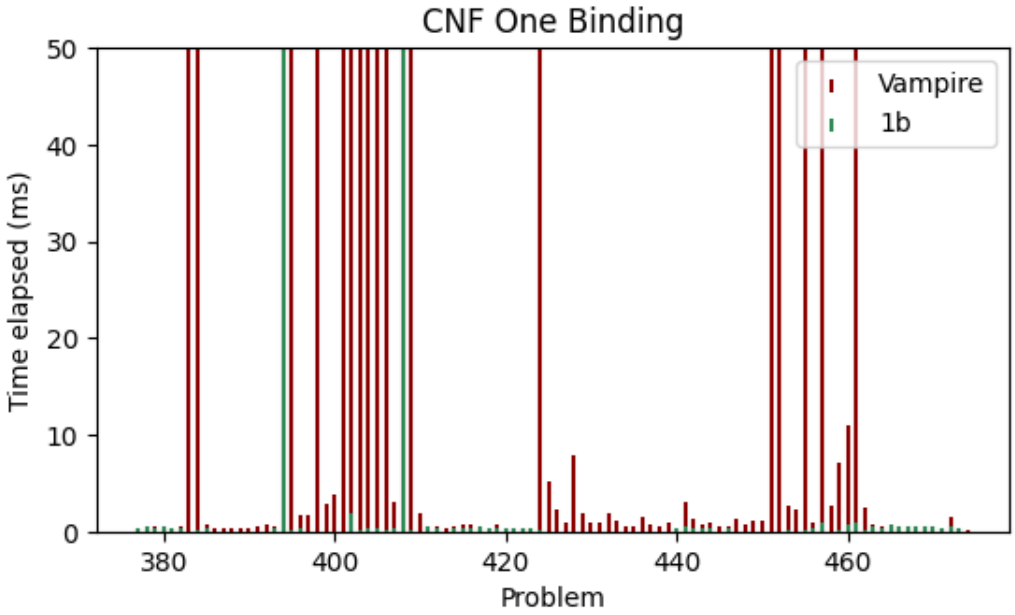

Figura 1.8: Tempo Vampire vs 1b, problemi cnf

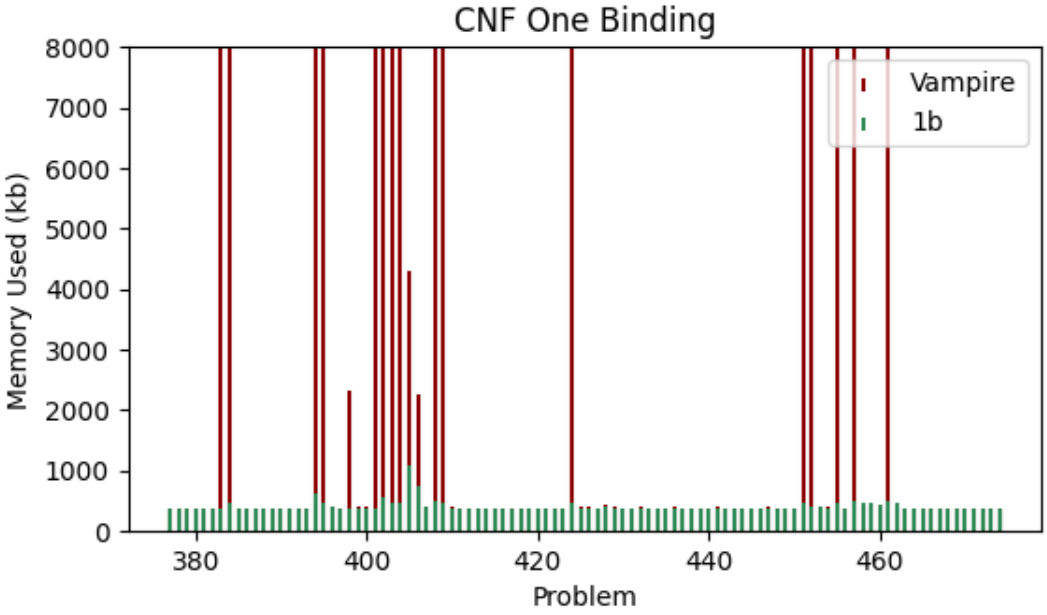

Figura 1.9: Memoria Vampire vs 1b, problemi cnf

1.3 Ottimizzazioni

1.4 Conclusioni e Possibili Sviluppi futuri